



National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

UNIVERSITY OF ALBERTA

Routing in Blocking and Non-Blocking Multi-Stage
Networks for Parallel Systems

BY

Chin-Hung Lam



A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

DEPARTMENT OF COMPUTING SCIENCE

Edmonton, Alberta
Spring 1995



National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file - Votre référence

Our file - Notre référence

THE AUTHOR HAS GRANTED AN
IRREVOCABLE NON-EXCLUSIVE
LICENCE ALLOWING THE NATIONAL
LIBRARY OF CANADA TO
REPRODUCE, LOAN, DISTRIBUTE OR
SELL COPIES OF HIS/HER THESIS BY
ANY MEANS AND IN ANY FORM OR
FORMAT, MAKING THIS THESIS
AVAILABLE TO INTERESTED
PERSONS.

L'AUTEUR A ACCORDE UNE LICENCE
IRREVOCABLE ET NON EXCLUSIVE
PERMETTANT A LA BIBLIOTHEQUE
NATIONALE DU CANADA DE
REPRODUIRE, PRETER, DISTRIBUER
OU VENDRE DES COPIES DE SA
THESE DE QUELQUE MANIERE ET
SOUS QUELQUE FORME QUE CE SOIT
POUR METTRE DES EXEMPLAIRES DE
CETTE THESE A LA DISPOSITION DES
PERSONNE INTERESSEES.

THE AUTHOR RETAINS OWNERSHIP
OF THE COPYRIGHT IN HIS/HER
THESIS. NEITHER THE THESIS NOR
SUBSTANTIAL EXTRACTS FROM IT
MAY BE PRINTED OR OTHERWISE
REPRODUCED WITHOUT HIS/HER
PERMISSION.

L'AUTEUR CONSERVE LA PROPRIETE
DU DROIT D'AUTEUR QUI PROTEGE
SA THESE. NI LA THESE NI DES
EXTRAITS SUBSTANTIELS DE CELLE-
CI NE DOIVENT ETRE IMPRIMES OU
AUTREMENT REPRODUITS SANS SON
AUTORISATION.

ISBN 0-612-01715-X

Canada

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Chin-Hung Lam

TITLE OF THESIS: Routing in Blocking and Non-Blocking Multi-Stage Networks for Parallel Systems

DEGREE: Doctor of Philosophy

YEAR THIS DEGREE GRANTED: 1995

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

(Signed)

Chin-Hung Lam

Chin-Hung Lam
#3A-8915, 112 Street
Edmonton, Alberta
T6G 2C5 Canada

Date:

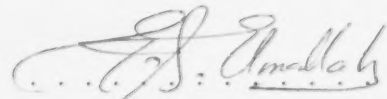
Jan 30, 1995

"The greatest happiness you can have is knowing that
you do not necessarily require happiness."
– William Saroyan

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

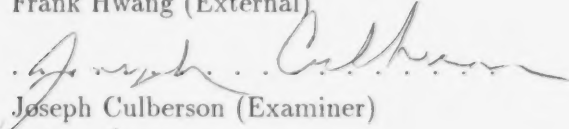
The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Routing in Blocking and Non-Blocking Multi-Stage Networks for Parallel Systems** submitted by Chin-Hung Lam in partial fulfillment of the requirements for the degree of Doctor of Philosophy.



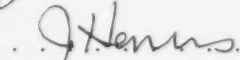
Ehab Elmallah (Supervisor)

..... 

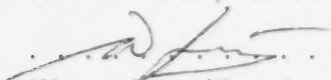
Frank Hwang (External)



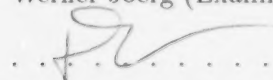
Joseph Culberson (Examiner)



Janelle Harms (Examiner)



Werner Joerg (Examiner)



Pawel Gburzynski (Examiner)

Duane Szafron (Chair)

Date: *Dec. 14, 1994*

To my grandparents

Abstract

Interconnection networks (INs) are critical components of high performance parallel and distributed systems. Multi-stage INs (MINs) are made of links and switches that are arranged in stages. MINs are attractive because of their superior scalability, regularity, and cost/performance ratio. However, many routing problems, mostly related to multi-path MINs, are still open. We study routing problems on four recursively decomposable networks that are of practical and theoretical interest.

After adding k extra stages to a Generalized Cube Network (GCN), 2^k paths become available to each input/output pair of the resulting network, called the k -GCN. Useful symmetry properties of the k -GCN are shown. Given a traffic matrix and a 1-GCN, we prove that using the 2 alternative paths between any input/output pair with equal probabilities minimizes the average packet delay, under a wide range of practical traffic patterns and some restrictions. A simulation study that verifies our probabilistic analysis is presented.

We next consider the class of Augmented Data Manipulator (ADM) networks. ADM networks can realize many useful permutations but no known algorithm can efficiently identify and route all admissible permutations. By exploiting the regularity of the topology, we devise a deterministic, backtracking algorithm for routing permutations on the ADM network at compile time. The algorithm requires $O(N^3)$ time in the worst case and its average time complexity is almost linear, as indicated by simulation. An upper bound on the average time complexity is also derived.

High speed multiprocessor systems demand nonblocking networks, but cost considerations typically force the architect to adhere to rearrangeable or blocking MINs.

The notion of a k -nonblocking network is introduced to provide a framework for bridging the gap between blocking and nonblocking networks with a spectrum of new networks. k -nonblockingness conveniently quantifies the power of MINs, especially under unspecified traffic. A systematic scheme for constructing k -nonblocking networks is presented; together with efficient parallel and sequential routing algorithms. When $k = N$, our construction gives a network cheaper than an existing strictly nonblocking network.

Define the width of a MIN to be the maximum number of links running between any 2 adjacent stages. We show that all strictly nonblocking MINs of size N must have width $> N$. The lower bound on the width of wide-sense nonblocking MINs was unknown, except for $N = 4$. By proving that a new network, called the *Quadruplet*, is wide-sense nonblocking, we conclude that N is a tight lower bound on the width of wide-sense nonblocking MINs. By cascading a sufficient number of any rearrangeable MINs of width N , a wide-sense nonblocking MIN can be obtained. An upper bound on the number of shuffle/exchange stages needed to attain wide-sense nonblockingness is also derived. Finally, a new implication of the notion of universality is shown.

Preface

This dissertation documents part of the results obtained during my study in the department of Computing Science at the University of Alberta. Each of Chapters 3 to 6 concentrates on a sub-topic. They are roughly arranged in the order of increasing depth. (Interestingly, the weight of mathematics decreases while the weight of computer science increases in that order.) As these sub-topics are "loosely-coupled," Chapters 3 to 6 can be read in any order, although Chapter 6 refers to the k -nonblocking network concept introduced in Chapter 5. The four sub-topics reveal different facets of a central issue in the study of MINs, namely routing, which is the main theme of this dissertation. The emphasis is on routing problems in multi-path MINs, especially those that allow recursive decomposition. It is hoped that our efforts would help build a unified theory of routing in MINs.

Preliminary versions of Chapters 3, 4, and 5 have appeared in conference proceedings.

Acknowledgements

I received two research topics, which correspond to Chapters 3 and 4 of this dissertation, from my supervisor Dr. Ehab Elmallah. As members of my supervisory committee, Dr. Joseph Culberson and Dr. Janelle Harms are my major sources of constructive comments at all stages of this research. I am in debt to other members of my examination committee, including Dr. Frank Hwang from AT&T and Prof. Werner Joerg, for their careful reading of drafts of this dissertation. Dr. Anup Basu kindly served on my candidacy examination committee. Dr. Christopher Smyth at the University of Edinburgh reviewed drafts of Chapters 2, 5 and 6 and gave useful suggestions (and I should also thank Prof. Nicholas Pippenger at the University of British Columbia for putting me in contact with Dr. Smyth). Discussions with my fellow graduate students has always been rewarding.

This work is partially supported by NSERC Canada under grant number OGP 36899. The Department of Computing Science at the University of Alberta also financed the author through graduate assistantships.

My stay in Edmonton would not have been an enjoyable one without the company of good friends such as Alice Chan, Ben Wong, and Clement Kwok.

Contents

1	Introduction	1
1.1	Why Multi-Stage Interconnection Networks?	1
1.2	Architectural Models	3
1.3	Routing Problems	4
1.4	Overview of Our Research	5
1.5	Organization of This Thesis	9
2	Preliminaries	10
2.1	Notation and Terminology	10
2.2	Switch and Network Models	11
2.3	Classification	13
2.3.1	Beneš' Classification	13
2.3.2	Feng's Classification	15
2.4	Some Existing Networks	17
2.4.1	Clos Network	17
2.4.2	The Shuffle-Exchange Network	18
2.4.3	Cube-Type Networks	19
2.4.4	Beneš' Rearrangeable Network	23
2.4.5	The Data Manipulator Network	24
2.5	Equivalence of Networks	25
2.6	Figures of Merits	26
2.7	Complexity of Permutation Routing	27

3	Routing on Extra Stage Networks	32
3.1	Introduction	32
3.2	Switch Model and Notation	35
3.3	Properties of the k -GCN	36
3.4	Load Balancing on the 1-GCN	39
3.4.1	Basic Concepts and Definitions	40
3.4.2	Probability Assignment	41
3.5	Simulation Study	44
3.5.1	Experiment and Simulator Design	44
3.5.2	Results and Interpretations	45
3.6	Concluding Remarks	50
4	Permutation Routing on the ADM Network	54
4.1	Introduction	54
4.2	The Network, the Notation, and the Problem	56
4.3	Routing on the ADM	59
4.3.1	Routing One Connection Request	59
4.3.2	Permutation Routing	60
4.3.3	The Routing Algorithm	64
4.4	Simulation Experiment	67
4.4.1	Program Verification	68
4.4.2	Performance Study	69
4.5	Approximate Time Complexity Analysis	72
4.5.1	Classification of Permutations	72
4.5.2	Cardinality of W_i^*	74
4.5.3	Number of Nodes Searched	77
4.5.4	An Upper Bound	79
4.5.5	Worst-Case Performance	81
4.6	Conclusion	81

5	On a Class of k-Nonblocking Networks	84
5.1	Introduction	85
5.2	k -Nonblockingness	86
5.2.1	Definition	86
5.2.2	k -Nonblockingness of Some Existing Networks	87
5.3	Constructing a k -Nonblocking Network	89
5.3.1	Interpretation and Application	93
5.3.2	A Special Case: N -Nonblocking	94
5.4	Routing Algorithms	95
5.4.1	MLP	95
5.4.2	FirstFit	96
5.5	Removing Dummy Switches	98
5.6	Conclusion	99
6	Width N Wide-Sense Nonblocking Networks	101
6.1	Introduction	101
6.2	Width of Strictly Nonblocking MINs	105
6.3	Definitions and Notations	105
6.4	Basic Properties of the Q Network	107
6.5	The Q Network is Wide-Sense Nonblocking	111
6.6	More About Q_N	112
6.7	Transforming Q_N into a Proper MIN	114
6.8	Advanced Results	118
6.9	Concluding Remarks	120
7	Conclusion	122
	Bibliography	124
A	Permutation Routing Complexity: 3-stage INs	135

B Row Data from the Simulation of the 1-GCN	138
C Maple Source Code for Computing $\widetilde{\mathcal{S}}_N$	145

List of Figures

1.1	A simple classification of INs.	2
1.2	Two multiprocessor configurations: a) PE-PE, b) P-M.	4
1.3	Three key elements in the study of INs.	5
1.4	Scopes of the 4 research sub-topics.	6
2.1	States of a 2x2 crossbar switch.	12
2.2	Graph models for MINs; a) switches as nodes, b) links as nodes. . . .	12
2.3	Beneš' classification scheme.	15
2.4	Two alternative paths in a 3x3 crossbar network.	15
2.5	Topologies of INs under Feng's classification.	17
2.6	Three-stage Clos network $v(m, n, r)$	17
2.7	A shuffle-exchange network of size 8 (a) and its graph model (b). . . .	19
2.8	An Omega network of size 8.	20
2.9	The baseline network (b) and its recursive model (a).	21
2.10	A GCN of size 8 (a) and its corresponding 3-cube (b).	22
2.11	A size 8 GCN serving a request from input 2 to output 4.	23
2.12	An ESC network of size 8.	23
2.13	Beneš' rearrangeable network (b) and its recursive model (a).	24
2.14	An 8-node PM2I network (a) and the corresponding DM network (b). . .	25
2.15	Special 2x2 switch allowing only one connection.	28
2.16	Constructing the MIN for (a) the variables, (b) the clauses.	29
2.17	An example for $c_j = \bar{x}_1 + x_2 + x_n$	30

3.1	1-GCN and 2-GCN of size 8.	33
3.2	2x2 switch with unlimited buffer queues.	35
3.3	Partitioning switches in a 2-GCN of size 16 into 4 classes.	37
3.4	The scenario in which a conflict may occur.	42
3.5	Effect of workload (w) on queue lengths for $N = 8$	47
3.6	Queue lengths at various stages (s) for $N = 8$	48
3.7	Effect of workload (w) on queue lengths for $N = 16$	49
3.8	Queue lengths at various stages (s) for $N = 16$	51
3.9	Screen dump of simulation run that contains a hotspot at output 10.	52
4.1	An ADM of size 16 (redundant links in stage 4 removed for simplicity).	57
4.2	A size 8 ADM split into even and odd sub-networks.	58
4.3	a) A component, b) 2 components, c) p_- , d) p_+ , e) p_r , f) p_i	58
4.4	Intermediate permutations for a given admissible permutation.	59
4.5	a) Case p_+ in A_1 . b-e) Various scenarios in A_1	61
4.6	Worst case search tree.	66
4.7	Average (a) and maximum (b) number of nodes searched per permutation.	70
4.8	Average time to process each permutation; logarithmic and linear scale.	71
4.9	A classification of permutations for $N = 16$	74
4.10	A v-case switch in stage 0.	75
4.11	Bipartite graphs showing stage 0 and n	76
4.12	Explanation for the terms in T_l	78
4.13	$\widetilde{\mathcal{S}}_N$ and $\lg N$ versus N	80
5.1	Beneš' network. Existing request: (i, j) . New request: (i', j')	87
5.2	The third request $(6, 1)$ is blocked in Beneš network.	88
5.3	Highlighted switches in $C(4, 8)$ are unusable by any path from i to j	91
5.4	Constructing an exactly 3-nonblocking network of size N	94
5.5	Three bit encoding of the state of a switch.	97

6.1	Q_N .	106
6.2	The first quarter of a Q_5 .	106
6.3	a) Terminal labeling of a Q_N , b) Busy path (a, a') removed from graph.	107
6.4	Network graphs resulting from the 2 ways to realize (a, b') .	108
6.5	Network graphs resulting from the 2 ways to realize (b, b') .	108
6.6	a) BPPs (a, b') and (b, a') . b) $N-2$ (b, b') requests using their LPPs.	110
6.7	State transition diagram for Q_N .	111
6.8	After removing 1 stage from Q_N .	113
6.9	Beneš' size 4 wide-sense nonblocking network.	113
6.10	Using dummy switches to eliminate links that bypass stages in Q_4 .	115
6.11	The twist in a) involves 2 switches while b) involves 4. i and j are integers, where $i \geq 1$ and $0 < j < m$.	116
6.12	Elimination of multiple connections using 2 types of switches.	116
6.13	Elimination of multiple connections using 1 type of switches.	117
6.14	An extended Clos network.	119
6.15	SE as a building block for width N wide-sense nonblocking MINs.	119
A.1	Possible cases for a request (i, j) .	136

List of Tables

4.1	Number of ADM-admissible permutations: Leland's versus our results.	69
4.2	Numerical results produced by Maple V.	79
4.3	Comparison of the ADM and Ω networks.	82
4.4	$N!$, $(N!)^{0.694}$, and $(N!)^{0.5}$ versus N	82
5.1	m verses $k_{\max}$	93
5.2	Minimum depth necessary to be k -nonblocking.	100
6.1	Minimum depth required for width N MINs of various capabilities. .	102
6.2	Some results in the study of iterated MINs.	103
6.3	Priority assignment for the 4 types of requests.	109
6.4	Switch(es) shared by any 2 requests.	109
7.1	An overview of new results (marked by *) in Chapters 5 and 6. . . .	123
B.1	Uniform traffic: average queue length (over 4 runs) at each stage. . .	139
B.2	Uniform traffic and $N = 8$: average queue length, variance, and confidence interval.	140
B.3	Uniform traffic and $N = 16$: average queue length, variance, and confidence interval.	141
B.4	Hotspot traffic: average queue length (over 4 runs) at each stage. . .	142
B.5	Hotspot traffic and $N = 8$: average queue length, variance, and confidence interval.	143

B.6 Hotspot traffic and $N = 16$: average queue length, variance, and confidence interval.	144
---	-----

Chapter 1

Introduction

This chapter first gives motivations for the study of multi-stage interconnection networks (MINs) in § 1.1. The role of a MIN in a multiprocessor system is examined in § 1.2. In § 1.3, we highlight the significance of a central issue: routing in MINs, especially multi-path MINs that are recursively decomposable. An overview of our research is given in § 1.4. The organization of this dissertation is summarized in § 1.5.

1.1 Why Multi-Stage Interconnection Networks?

Efficient transfer of data between communicating parties is a major challenge shared by telecommunication and multiprocessor systems [46, 48]. The design of effective *interconnection networks* (INs) proves to be a critical part of such high performance systems (e.g., [88]). Although we will concentrate on INs for multiprocessor systems, note that many INs serve telecommunication and multiprocessor systems equally well. For instance, the *shared-bus* has been widely used on local area networks as well as multiprocessor systems. The *star* topology is another example. With the advent of distributed computing, the border between communication and computation has become blurred. In fact, the study of INs is rooted in research in telephony [10, 24].

We now examine the role of multi-stage INs (MINs) using the simple classification tree in Figure 1.1. In *direct* networks (a.k.a. *static* [41] or *point-to-point* networks),

communicating parties are directly connected by *links*. Examples include the star and the hypercube network [26, 41, 51] (comparative simulation studies can be found in [104, 105]). Special graphs [47] and groups [35] have been proposed as directed networks. In *indirect* networks (a.k.a. *dynamic* networks [41]), *switches* are employed such that paths between communicating parties involve alternating links and switches.

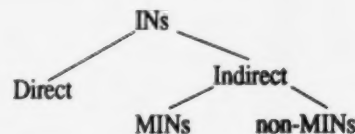


Figure 1.1: A simple classification of INs.

If the switches in an indirect network can be arranged into *stages* such that switches in one stage are only connected to switches in subsequent stages, then it is a MIN. Examples are given in Chapter 2. Switches in indirect networks that are not MINs (denoted *non-MINs*) cannot be arranged into stages. Existence of feedback loops (e.g., [74]) and mixing of traffic directions (e.g., [49]) are typical features in non-MINs. Simplicity, or ease of analysis, makes MINs more popular than non-MINs in the literature. Other attractive virtues of MINs include:

Scalability: It is relatively easy to expand an existing MIN to accommodate a heavier workload (e.g., a larger number of terminals). In the worst case, the expansion process requires redesigning the entire backbone network or its supporting software component. In this regard, MINs usually exhibit enough structural symmetry to make the expansion task relatively simple.

Modularity/regularity: MINs can be implemented as separate modules in the system. They can be upgraded/replaced/repared independently of other components.

Cost/performance ratio: The cost of the switches in MINs is justified by the large increase in power (measured by, say, connection power¹ or bandwidth).

¹The term connection power generally means the capability of satisfying arbitrary 1-to-1 requests.

MINs have been used in a number of multiprocessor systems. For instance, the butterfly network is used in the BBN system [107]. A number of multi-stage cube networks have been used in or proposed for real systems [114]. It is expected that MINs will remain a critical component of high performance multiprocessor systems, especially those that operating in SIMD (single-instruction-stream-multiple-data-stream) mode. MINs have been proposed for high speed telecommunication systems, such as integrated services digital networks (ISDNs). See [4, 121] for good surveys. MINs are also suitable for ATM (Asynchronous Transfer Mode) switching [91].

Some MINs have corresponding direct networks. For instance, the generalized cube network (GCN) [112, Ch. 5] and the data manipulator network [40] are multi-stage versions of the direct networks hypercube and plus-minus-2- i (PM2I) [41], respectively. For a brief introduction to the study of MINs see [6, 113].

1.2 Architectural Models

This section examines the role of an IN as the primary communication infrastructure in multiprocessor systems. Two typical configurations are considered [18, 56, 112]:

Processor-to-memory (P-M): A number of processors are connected to a number of memory units via the IN (Figure 1.2b). Since a party on either side of the IN never communicates with other parties on the same side, these INs are also called *2-sided* networks [20]. If a uni-directional IN (e.g., an IN operating in packet-switching mode) is used, then typically 2 identical INs are employed to facilitate duplex communication.

Processing-element-to-processing-element (PE-PE): A number of *processing elements* (PEs) are connected by an IN (Figure 1.2a). Each PE consists of a processor and local memory. Since every party can communicate with any other party, these INs are also called *1-sided* networks [20, 89] and they are understood to be bi-directional. However, a uni-directional 2-sided IN can serve as a 1-sided network if the i th input and the i th output are both connected to the i th PE.

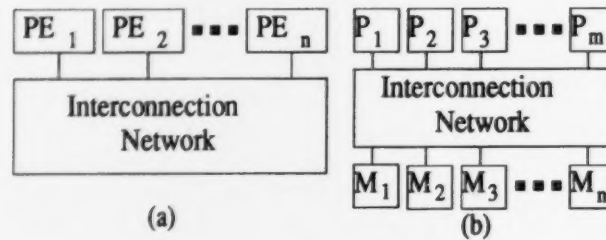


Figure 1.2: Two multiprocessor configurations: a) PE-PE, b) P-M.

See [112, Ch. 2] for more details and examples. Hybrid configurations (e.g., the partitionable SIMD/MIMD² system called PASM in [114]) evolved from the 2 basic configurations exist but they will not be covered. Some multiprocessor systems have a dedicated IN for cache coherency [83].

1.3 Routing Problems

In this section, motivations for the study of routing problems in MINs are presented. We believe that in the study of MINs, a long term goal shared by both computer scientists and engineers is the complete resolution of the relationship between 3 key elements: *network*, *routing*, and *traffic*. By traffic we broadly mean a permutation to be satisfied, or a set of 1-to-1 connections to be made (includes broadcasting), or a group of packets to be transmitted that satisfy some distribution. Loosely speaking, a *routing algorithm* controls the underlying hardware, using full knowledge of the network topology and current state, so that either the given traffic requirement is satisfied, or the traffic requirement is rejected as not satisfiable. Figure 1.3 depicts the central position of routing. A unified theory of the 3 key elements should be able to solve typical research problems such as:

- Given network A , what routing algorithm can satisfy traffic requirement C ?
- Given network A and routing algorithm B , what kind of traffic can be handled?

²MIMD means Multiple-Instruction-stream-Multiple-Data-stream.

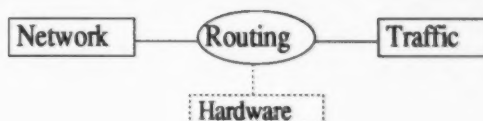


Figure 1.3: Three key elements in the study of INs.

- Given traffic requirement C , what network and routing algorithms are needed?
- Given a new network A , what kind of traffic can be handled and what routing algorithms are applicable?

Now, the central role of routing in the unified theory becomes clear. Note that some routing problems are intractable (e.g., permutation routing on arbitrary MINs is an NP-complete problem as shown in § 2.7). We should identify such cases, and accordingly focus more on tractable routing problems that arise in practice. To construct such a unified theory, we not only need to study the routing algorithms for individual networks and the kind of traffic they can handle, we also need to generalize existing network models and routing algorithms to cover a larger class of networks. These 2 directions can be pursued simultaneously. Chapter 3 and 4 of this dissertation study routing problems in individual networks while Chapter 5 and 6 attempt to generalize network models and routing algorithms. *Unique-path* MINs have been studied extensively in the literature; so this dissertation focuses on *multi-path* MINs, especially those that allow recursive decomposition. Therefore, although our 4 sub-topics seem to be loosely-coupled and have different depths, routing on recursively decomposable multi-path MINs forms the backbone of this dissertation.

The importance of routing has been recognized in the literature. [75] contains a comprehensive survey of the subject and a substantial bibliography.

1.4 Overview of Our Research

Based on the above discussions, our research goal is to design, study, and generalize routing algorithms such that eventually a unified theory of routing can be derived.

This thesis emphasizes multi-path MINs that are recursively decomposable.

Although Beneš' classification [13] will not be discussed in detail until Chapter 2, it is convenient to illustrate the broad coverage of our work using his scheme. In order of increasing connection power, INs (of N inputs/outputs) are classified into:

Blocking (B) networks: unable to realize all $N!$ permutations,

Rearrangeable (R) networks: able to realize all $N!$ permutations,

Wide-sense nonblocking (W) networks: can realize any request/release sequence if path-selection is guided by a rule, and

Strict-sense nonblocking (S) networks: can realize any request/release sequence even when paths are randomly selected.

Figure 1.4 depicts the relationship between them. The coverage of the 4 sub-topics are labeled by its corresponding chapter numbers in the figure.

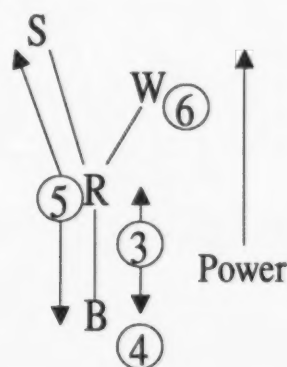


Figure 1.4: Scopes of the 4 research sub-topics.

Now, we briefly review each of the sub-topics:

Routing on Extra-Stage Networks (Chapter 3): This work deals with MINs with capabilities ranging from blocking to rearrangeable. Our study is motivated by a more general problem: given a MIN that provides multiple paths between its input and output terminals, and a certain traffic distribution, what packet routing strategy minimizes the average delay of the network? We focus on traffic

patterns that can be described by a matrix L , in which element l_{ij} represents the probability of generating a packet from input i to output j in a time unit. A specific network, the 1-GCN, formed by adding an extra stage to the input side of a Generalized Cube Network (GCN) [112], is considered. The 1-GCN provides 2 alternative paths between every input/output pair. By identifying some symmetry properties of the 1-GCN, we show that the average network delay is minimized when the 2 alternative paths are used with equal probabilities, regardless of the traffic patterns, under some restrictions. This argument is verified by simulation. The symmetry properties we identified generalize to k -GCNs, in which k extra stages are added to the input side of a GCN.

Permutation Routing on the ADM Network (Chapter 4): This work concentrates on a blocking MIN called the Augmented Data Manipulator (ADM) Network [112, Ch. 6]. No known algorithm can efficiently route permutations through the ADM [76]. The set of permutations admissible by the ADM have not been characterized. We devise a deterministic, backtracking algorithm for routing admissible permutations and rejecting inadmissible permutations. Although in the worst case, the algorithm takes $O(N^3)$, in most cases it apparently takes linear time. Simulation is used to demonstrate its good performance. An upper bound on its average time complexity is also derived. The algorithm is suitable for compile-time routing on SIMD machines.

On a Class of k -nonblocking Networks (Chapter 5): This work basically involves MINs with capabilities ranging from blocking, rearrangeable, to strict-sense nonblocking. Although nonblocking networks can satisfy the communication demand in high performance systems, their high complexity typically forces the architect to adhere to rearrangeable or even blocking networks. We introduce the notion of k -nonblocking networks as one that never blocks an incoming request whenever there are $< k$ existing connections in an N input network, where $1 \leq k \leq N$. On the theoretical side, the general concept of k -nonblockingness

bridges the complexity gap between blocking and nonblocking networks by a spectrum of new networks. It also provides a convenient way to quantify the connection power of a network under unspecified traffic. On the practical side, we present a systematic scheme for constructing k -nonblocking networks in the strict sense. Efficient parallel and sequential algorithms for the proposed networks are given. For the case $k = N$, the constructed network is cheaper than an existing nonblocking network [79] that is, like ours, based on Beneš' rearrangeable network [12, 11].

Wide-Sense Nonblocking Networks Made of Square Switches (Chapter 6): This work focuses on the class of wide-sense nonblocking networks. The following 2 related open problems have motivated this research:

1. The *width* of a MIN is the maximum number of links running between any 2 adjacent stages. It can be derived from [14] that the width of any strict-sense nonblocking MIN of N inputs/outputs must be $> N$. In contrast, the width of any rearrangeable MIN (e.g., Beneš' rearrangeable network [12, 11]) must be $\geq N$. Although Beneš has given a 4×4 wide-sense nonblocking MIN of width 4 in [14], it is not known whether wide-sense nonblocking MINs can generally have width N because Beneš' network does not generalize to $N > 4$.
2. Empirically, increasing the number of stages in a MIN increases its connection power (mentioned briefly in [61]). For instance, a single stage shuffle-exchange (SE) network [117] is very limited in connection power. But when there are $\log_2 N$ SE stages, the resulting Omega (Ω) network [73] supports many useful permutations although it is still a blocking network. As the number of SE stages reaches $3 \log_2 N - 4$, the network becomes rearrangeable [80]. It is natural to ask what is the most powerful network obtainable by merely adding more stages to a MIN. The previous paragraph eliminates the possibility of obtaining a strict-sense nonblocking

MIN. The case for wide-sense nonblocking MINs is still open.

To declare a network as wide-sense nonblocking one must give, or at least prove the existence of, a routing algorithm that satisfies any request/release sequence. This is in general very difficult. We first propose a width N , recursive network called the Quadruple (Q). Then we prove that it is wide-sense nonblocking by giving a routing algorithm. Afterwards, we transform the Q network, which is not a proper MIN, into a proper one. The open problems are thus solved. A fundamental difference between wide- and strict-sense nonblocking MINs that is not obvious from their definitions is identified: the former can have width $= N$ while the latter must have width $> N$. We also show that by cascading a finite number of any rearrangeable MINs of width N , a wide-sense nonblocking MIN can be obtained. An *iterated* [43] MIN uses a single permutation repeatedly to connect all adjacent stages. We prove that the set of permutations out of which rearrangeable iterated MINs can be built is identical to the set of permutations out of which wide-sense nonblocking iterated MINs can be built.

1.5 Organization of This Thesis

Chapter 1 of this dissertation introduces the reader to the general area of MINs and routing problems. The emphasis is on multi-path MINs that are recursively decomposable. Chapter 2 provides some background information on MINs. We cover notations, terminologies, models, classification schemes, outstanding results, and complexity of permutation routing on arbitrary MINs. Material specific to each sub-topic is delayed to their respective chapters. Each of Chapter 3 to 6 discusses a particular sub-topic as mentioned before. Although they are roughly sorted by increasing depth, and Chapter 6 uses the k -nonblocking concept introduced in Chapter 5, these 4 chapters can be read in any order. Each chapter includes its own concluding remarks. Chapter 7 summarizes this dissertation and discusses some future directions.

Chapter 2

Preliminaries

This chapter presents some background information required throughout the thesis. Some notation and terminology are introduced in § 2.1, followed by a description of common switch and network models in § 2.2. Two well known network classification schemes are reviewed in § 2.3. Important results on five representative networks in the literature are highlighted in § 2.4. In § 2.5, we examine the problem of equivalence of networks. § 2.6 reviews the major factors to be considered in the design of MINs. In § 2.7, we show that the general problem of permutation routing on an arbitrary MIN is NP-complete.

2.1 Notation and Terminology

MINs are generally modeled by *graphs* or *digraphs* (directed graphs) so we will use standard graph and network terminology (see for example [21]). An N_i -by- N_o (denote $N_i \times N_o$) MIN has N_i *inputs* and N_o *outputs*. When $N_i = N_o = N$, we say the network has *size* N . A MIN is made of *switches* and *links* arranged into *stages*, as in Figure 2.8. All MINs can be modeled by directed acyclic graphs (DAGs). For ease of analysis, most MINs proposed are *layered*, in which switches in one stage are only connected to those in the next stage [67]. All MINs considered in this thesis are layered. Furthermore, most MINs have at most 1 link between any 2 switches in 2 adjacent stages. If there

are multiple physical links between some pairs of switches, it is called a *multi-network* [7]. A *dilated network* provides multiple links either logically or physically [118].

A *path* is an alternating sequence of links and switches that connect an input and an output. An input (output) is *idle* if it is not used in any existing connections. A path is *idle* if it can be allocated to an incoming request without disturbing existing connections, otherwise, it is *busy*. A MIN is *idle* if all the paths are idle.

The *depth* [39] is the maximum number of switches in any path in a non-layered MIN, or simply the number of stages in a layered MIN. The *cross-section width*, or *width* [8] for short, between 2 stages is the number of links running between them. The *width* of a MIN is the maximum width between any 2 stages. The *hardware cost*, or just *cost*, of a MIN is usually measured by the total number of switches, or more broadly, the total number of *cross-points* [10] if the switches are not uniform. By definition, an axb crossbar switch has ab cross-points.

We denote $\log_2 N$ by $\lg N$. $n = \lg N$ is the depth of size N cube-type networks (§ 2.4.3). The binary representation of an r -bit number b is $(b_{r-1} \dots b_1 b_0)$. Concerning traffic patterns, this thesis deals with 1-to-1 requests only and ignores broadcasting and multicasting. Moreover, all incoming requests are assumed to be valid; that is, both the input and output in a request are currently idle.

Many applications on SIMD machines involve alignment of data and processors. A set of simultaneous connections between memories and processors, corresponding to a 1-to-1 mapping of inputs to outputs, is called a *permutation*. *Permutation networks* are specifically designed to handle this kind of traffic [57, 60, 78].

2.2 Switch and Network Models

MINs are sometimes designed with 2x2 crossbar switches as basic building blocks. Each switch provides 2 states for 1-to-1 connections (Figure 2.1a) and 2 states for 1-to-2 connections (Figure 2.1b). The *straight* (a.k.a. *non-crossing*) state is usually the initial, or the “inactive,” state of a switch in permutation networks. The *swap*

(a.k.a. *crossing* or *exchange*) state is typically the “active” state. The *upper-* and *lower-broadcast* states are used in broadcast operations. (One variant of this model provides an additional state for 1-to-1 connections [128].)

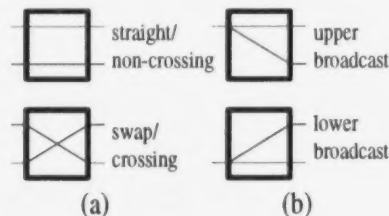


Figure 2.1: States of a 2x2 crossbar switch.

The above switch model is suitable for MINs operating in circuit-switching mode. In packet-switching mode, the model should be augmented to account for delays due to buffering and queuing. A number of models have been proposed for this purpose [81]. A typical one is used in Chapter 3. Basic tools for analyzing systems constructed from these models, under a given traffic distribution, include queuing theory [19, 29], petri net, mean value analysis (MVA) and approximate MVA [59, 126].

Two graph-theoretic models are frequently used for studying the topology of MINs. In the first model, *nodes* correspond to switches while *edges* correspond to links. Input and output terminals become nodes if they need to be modeled. Figure 2.2a shows the graph-theoretic model of a typical MIN under this scheme. In the second model,

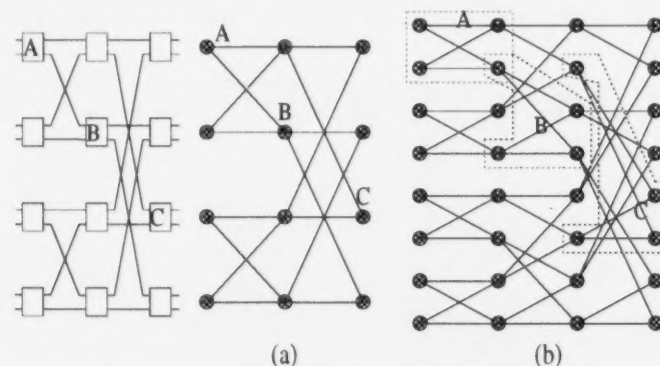


Figure 2.2: Graph models for MINs; a) switches as nodes, b) links as nodes.

a link becomes a node in the graph. Each crossbar switch is modeled by a group

of edges that fully interconnect the nodes, each of which corresponds to a link that is connected to the switch. Figure 2.2b exemplifies this second model. A typical application of these graph-theoretical models is for solving flow problems [34, 36]. Recently, group-theoretic models have been proposed for exploring the symmetry in the topologies of MINs [5].

2.3 Classification

Two IN classification schemes are reviewed below. (Other good classification schemes exist, such as that in [86].) Beneš' classification is suitable for theoretical studies while Feng's classification allows practical issues to be considered.

2.3.1 Beneš' Classification

Beneš' classification scheme [13] is concerned with the permutation power of INs. Although this scheme is extensible to broadcast traffic, this dissertation focuses on 1-to-1 requests. If we denote the number of permutations realizable by a size N network by S_N , then the *combinatorial power* (a.k.a. permutation power) of the network is $r = \frac{S_N}{N!}$, where $0 \leq r \leq 1$. In the order of increasing connection power, Beneš classifies INs into 3 categories:

Blocking networks have $r < 1$. Thus, some of the $N!$ permutations cannot be realized even when routing is done off-line (i.e., all N 1-to-1 connection requests are known at the time routing decisions are made). For on-line routing (i.e., a connection request is served immediately upon arrival, putting the network into a new state), some incoming requests cannot be accommodated because of conflicts with existing connections. In this case, the system is in a *blocking state* and the incoming request must wait until some existing connections are released. The Omega network [73] is an example.

Rearrangeable networks have $r = 1$, but the permutation must be known in advance (i.e., off-line routing). For on-line routing, any incoming request can be accommodated if the routing algorithm is allowed to rearrange existing connections when necessary. The system may reach a blocking state, but by rearranging existing connections it jumps to a *safe state* that can accommodate the request. Beneš' rearrangeable network [11, 12] is a classical example (Figure 2.12).

*Nonblocking networks*¹ have $r = 1$. Routing can be done either off-line or on-line. The system never enters a blocking state and rearrangement is not needed. Two sub-classes are identified:

Nonblocking networks in the wide sense: When multiple paths are available to an incoming request, a *path-selection algorithm* chooses one such that no blocking states can be reached. More details can be found in [14]. A 4x4 example network is in Figure 6.9. Interestingly, routing on this class of networks can be conceptualized as a 2-player (router/blocker) game [39].

Nonblocking networks in the strict sense: When multiple paths are available to an incoming request, any one can be chosen randomly. That is, there is no blocking state. Examples can be found in [79, 109] and Chapter 5.

Figure 2.3 illustrates Beneš' classification. It is worthwhile to mention the notion of *self-routing* networks, which is not in Beneš' classification. In a self-routing network, a message can be routed to its target once the destination address is given. For instance, the Omega network [73] is self-routing (Figure 2.8). If the network is not self-routing, the routing algorithm needs to know the state of the network to decide the path. Beneš' rearrangeable network is an example (Figure 2.12) [102].

Furthermore, a *standard-path network* (SPN) [49] is a self-routing network in which the standard path (determined by source and target addresses) is always idle, provided that all existing connections are using their standard paths. A crossbar [134] is a

¹Note that some authors, e.g. [20], use the term "nonblocking networks" to refer to all permutation networks that have $r = 1$. Such networks may only be rearrangeable under Beneš' classification.

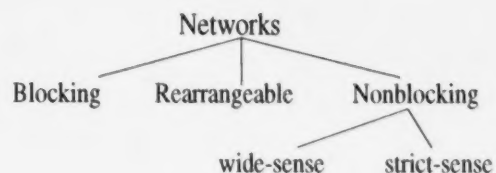


Figure 2.3: Beneš' classification scheme.

wide-sense nonblocking SPN. Figure 2.4 exemplifies the reason. Two of the many alternative paths between input 2 and output 3 in an idle 3x3 crossbar are shown. If path B is randomly chosen, then clearly some subsequent requests will be blocked. So the crossbar is not strictly nonblocking. By using path A, no subsequent requests will be blocked. In general, a request to connect input i and output j can be satisfied by activating cross-point (i, j) , so the path can be computed without considering the current state of the network.

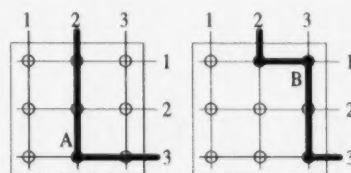


Figure 2.4: Two alternative paths in a 3x3 crossbar network.

2.3.2 Feng's Classification

While Beneš's classification concentrates on permutation power, Feng takes design issues into account [41]. In his classification, the cross-product of 4 key design decisions, namely operation mode, control strategy, switching methodology, and network topology, form a space of INs. Now we discuss the 4 decisions:

Operation mode: *Synchronous* communication is suitable for SIMD machines while *asynchronous* communication is suitable for MIMD (multiple-instruction-stream-multiple-data-stream) machines and ISDN systems. A combination of these 2 modes is also a viable alternative.

Control strategy: In *centralized control*, switch states can be set either by signals from a central controller or by retrieving bits from the globally computed routing tag that guides a packet through the network. In *distributed control*, each switch determines its action based solely on the source/destination addresses in incoming packets, without consulting other switches or packets.

Switching methodology: *Circuit-switching* [61, 84] is suitable for bulk data transmission while *packet-switching* [100, 121] is efficient for short messages. *Integrated-switching* combines these 2 techniques. Also, circuit-switching and packet-switching can simulate each other. Recently, *wormhole routing*, which attempts to incorporate the advantages of circuit- and packet-switching, has gained popularity (e.g., [33]). *Circuit-switched fixed routing* [135], which means using a simple heuristic rule to select paths, is another new alternative.

Network topology: This is a key factor in determining a suitable architectural structure. The numerous topologies proposed can be categorized into the tree structure shown in Figure 2.5. *Irregular* topologies (e.g., [9]) are common for telecommunication systems while *regular* topologies are popular with multiprocessor systems. Examples of 1-dimensional, 2-dimensional, and 3-dimensional static networks are the *ring*, the *mesh*, and the *binary 3-cube* network (Figure 2.10b), respectively. A single stage dynamic network is also called a *recirculating* network because several passes through the network may be necessary before a data item reaches its target. The shuffle-exchange network (§ 2.4.2) is an example.

2.4 Some Existing Networks

This section reviews some MINs that have been studied extensively in the literature. Good surveys of more INs can be found in [4, 20, 41, 86, 121].

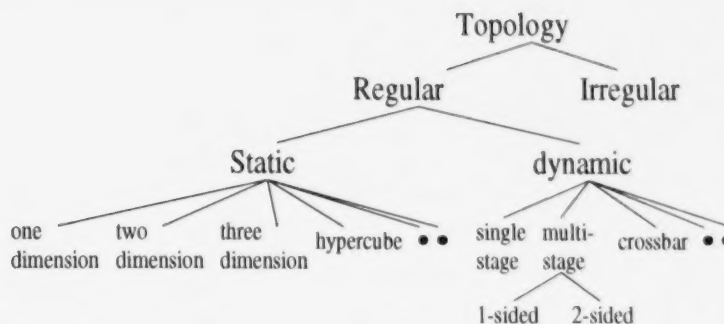
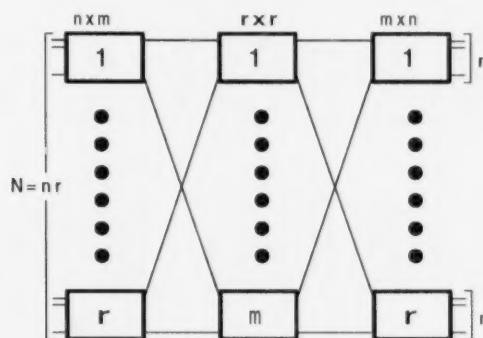


Figure 2.5: Topologies of INs under Feng's classification.

2.4.1 Clos Network

Clos introduced a 3-stage MIN in [24]. It is the basis of many other well known networks such as Cantor's network [23] and Beneš' rearrangeable network [11, 12]. A size $N = nr$ Clos network is denoted by $v(m, n, r)$, where m is a parameter indicating the number of rxr switches used in the middle stage (Figure 2.6).

Figure 2.6: Three-stage Clos network $v(m, n, r)$.

Every switch in the input stage is connected to every switch in the middle stage by a link. This gives a *full-access* pattern. The interconnection between the middle and output stages mirrors that between the input and middle stages. There are m alternative paths, each corresponding to a distinct switch in the middle stage, between every input/output pair. Theorem 2.1 relates m to the power of the network:

Theorem 2.1 [24]: *The 3-stage Clos network $v(m, n, r)$ is rearrangeable for $m \geq n$, and is strictly nonblocking for $m \geq 2n - 1$.*

The permutation routing problem on a rearrangeable Clos network is equivalent to an m -coloring problem on a bipartite multigraph. Given a permutation π to be routed on a $v(m, n, r)$ network, construct a bipartite multigraph $G = (V, E)$ where $V = \{u_1, u_2, \dots, u_r\} \cup \{v_1, v_2, \dots, v_r\}$. For each request (i, j) in the permutation, add an edge $(u_{\lceil i/n \rceil}, v_{\lceil j/n \rceil})$ to E . Then, an m -coloring of G corresponds to a feasible switch setting for realizing π . This equivalence to coloring problems extends to a larger class of networks that contains the Clos [64, 78].

A variants of Clos's network can be found in [32].

2.4.2 The Shuffle-Exchange Network

This important network has received much attention since its introduction by Stone [117]. It is the basic building block for many MINs such as the Omega network [73]. Define the *perfect shuffle* and *exchange* functions of an n -bit number $(b_{n-1} \dots b_0)$ as:

$$\begin{aligned} \text{shuffle}(b_{n-1} \dots b_0) &= (b_{n-2} \dots b_0 b_{n-1}) \text{ and} \\ \text{exchange}(b_{n-1} \dots b_0) &= (b_{n-1} \dots \overline{b_0}). \end{aligned}$$

A size $N = 2^n$ *shuffle-exchange* (SE) network (Figure 2.7a) is formed by cascading a stage of links implementing the shuffle function (on the terminal labels) with a stage of $\frac{N}{2}$ 2×2 switches for implementing the exchange function. Each switch performs an *identity* function if it is set to the *straight* state and performs an exchange function when it is in the *swap* state. Every switch is controlled independently. Figure 2.7b shows a graph-theoretic model for the SE network.

The permutation power of the SE network is very limited, but as shown in [117] it can perform many useful permutations when used repeatedly. The SE topology can easily be generalized to all even N ($\neq 2^n$). Several other generalizations of the original SE network can be found in the literature (e.g., [68]).

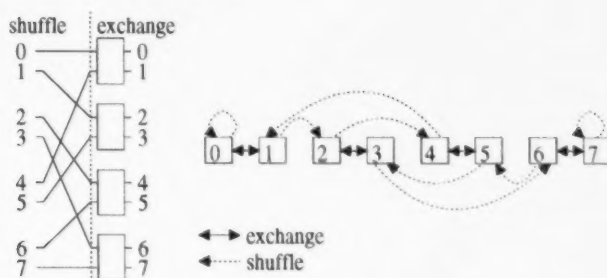


Figure 2.7: A shuffle-exchange network of size 8 (a) and its graph model (b).

2.4.3 Cube-Type Networks

The Generalized Cube Network (GCN) proposed by Siegel [112, Ch. 5] represents a class of topologically equivalent (§ 2.5) networks including the GCN, the Omega, the flip, the indirect binary n -cube, the baseline, the regular SW-banyan ($S=F=2$), and their inverse networks. These networks are now collectively known as *cube-type* or *hypercubic* networks. They have been used in several production systems [114].

Each of the cube-type networks has $n = \lg N$ stages. There are $\frac{N}{2}$ 2×2 crossbar switches in each stage. From any given input, there is exactly one path to any given output. This is called the *banyan* property [67]. Since each network can realize only $2^{\frac{N}{2} \lg N}$ distinct permutations, which is less than $N!$ for all $N \geq 4$, cube-type networks are self-routing, blocking networks.

Networks in this class share many properties. However, it is still instructive to examine individual networks because they are derived in quite different ways and they accept different sets of permutations (although the total number of acceptable permutations are the same). Below we review the Omega network, the baseline network, and the GCN.

The Omega Network

The Omega (Ω) network is proposed by Lawrie [73] as an alternative to using the single stage SE network iteratively [117]. Having $\lg N$ SE stages (Figure 2.8), the Ω provides *full-access* capability (any output can be reached from any input in 1 pass)

at optimal cost. It is an *iterated* network [43] because one single permutation is used between all stages. The modularized structure of the Ω simplifies its fabrication.

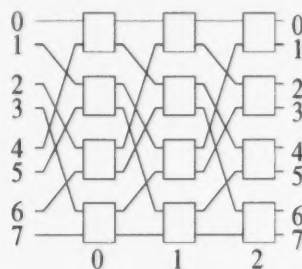


Figure 2.8: An Omega network of size 8.

The Ω network can realize many useful² permutations. A hierarchical classification of the permutations admissible by Ω -type networks is presented in [27]. Routing schemes and other properties can be found in [73]. A formal description of the Ω topology follows:

1. the inputs to each stage are labeled 0 to $N - 1$ from top to bottom. Thus, each switch has a pair of inputs of the form $(x, exchange(x))$ for $x \in \{0, \dots, N - 1\}$,
2. the label of the upper (lower) output of each switch is identical to the label of its upper (lower) input, and
3. stages are numbered from left to right starting from 0. Output y of stage k , is connected to input $shuffle(y)$ of stage $k + 1$, where $0 \leq k < \lg N$.

The Baseline Network

While the Ω is an iterated network, the baseline proposed by Wu and Feng [128] is a recursive network. It is constructed by interconnecting a stage of $\frac{N}{2} \times 2$ switches and $2 \times \frac{N}{2} \times \frac{N}{2}$ sub-networks by an inverse perfect shuffle (Figure 2.9a). The fully expanded network is shown in Figure 2.9b. An efficient routing algorithm can be found in [128].

²E.g., useful for the manipulation of matrices and computing fast Fourier transforms [73].

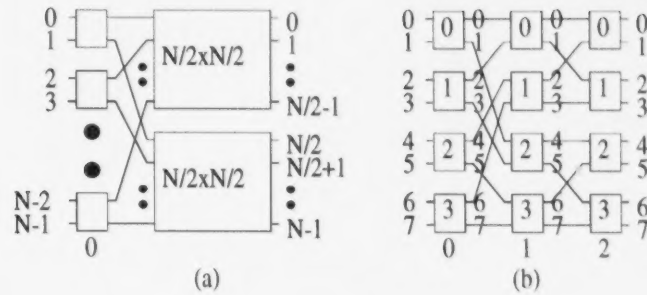


Figure 2.9: The baseline network (b) and its recursive model (a).

The GCN

The GCN is proposed by Siegel [112, Ch. 5] as a multi-stage version of the well known hypercube network. Let the i th binary cube function be $\text{cube}_i(b_{n-1} \dots b_0) = (b_{n-1} \dots \bar{b}_i \dots b_0)$. The GCN topology can then be described as follows:

1. the label of the upper (lower) output of each switch is identical to the label of its upper (lower) input,
2. the two inputs of a typical switch at the i th stage, where $0 \leq i \leq n-1$, has the form $(x, \text{cube}_i(x))$ for $x \in \{0, \dots, N-1\}$.

Note that a perfect shuffle pattern arises on the input side of the network from the above 2 rules. The i th stage simulates data transfers along the i th dimension of a hypercube (Figure 2.10). The inverse GCN (IGCN), which is a GCN traversed in reverse, is equivalent to the indirect binary n -cube network introduced by Pease III [97]. He also gives a remarkable characterization of the permutations *admissible* (realizable in 1 pass) by this network: Let P be a permutation on the set $\{0, 1, \dots, N-1\}$ of labels that maps an input address $X = (x_1 \dots x_n)$, where $n = \lg N$, to output address $Y = (y_1 \dots y_n)$. Now P can be described by n functions $f_1, f_2, \dots, f_n$ such that $y_i = f_i(x_1, \dots, x_n)$ for $i = 1, 2, \dots, n$.

Theorem 2.2 [97]: *A permutation P that maps X to Y is admissible by the indirect binary n -cube network if and only if $y_i = x_i + f_i(y_1, \dots, y_{i-1}, x_{i+1}, \dots, x_n)$ for $1 \leq i \leq n$. Modulo 2 arithmetic is used. (i.e., $+$ $\equiv \oplus$)*

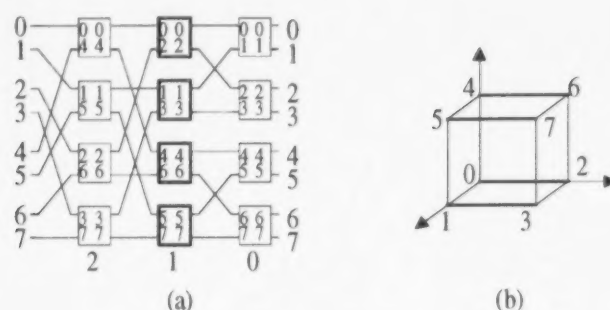


Figure 2.10: A GCN of size 8 (a) and its corresponding 3-cube (b).

Composition of networks is defined before we present the next theorem.

Definition 2.1 Let G and H be networks of size N . A new network denoted $G \cdot H$ is obtained by connecting output i of G to input i of H for $i = 0, 1, \dots, N - 1$.

Theorem 2.3 [97]: The composite network $\text{GCN} \cdot \text{GCN}^{-1}$ is rearrangeable, where GCN^{-1} is the inverse of GCN .

Theorem 2.3 has been used to prove the following important result:

Theorem 2.4 [97]: The composite network $\text{GCN} \cdot \text{GCN} \cdot \text{GCN}$ is rearrangeable for $\lg N \geq 3$.

A generalization of Pease III's work is in [106]. At least 2 well known routing tag schemes³ for the GCN exist [112, Ch. 5]:

1. *Exclusive-Or Tags*: To go from a source $S = (s_{n-1} \dots s_0)$ to a destination $D = (d_{n-1} \dots d_0)$ of a size $N = 2^n$ GCN, the switch in stage i in the path from S to D must be set to swap if $s_i \neq d_i$, and to straight if $s_i = d_i$. For instance, if $s_2s_1s_0 = 010$ and $d_2d_1d_0 = 100$, then $S \oplus D = 110$. The switches should be set to swap, swap, and straight (Figure 2.11).
2. *Destination Tags*: Regardless of the source, a packet can reach its destination $D = (d_{n-1} \dots d_0)$ if the packet leaves a switch in stage i using the upper (lower) output when $d_i = 0$ ($d_i = 1$).

³A routing tag scheme computes routing tags, which guide the packets through the network.

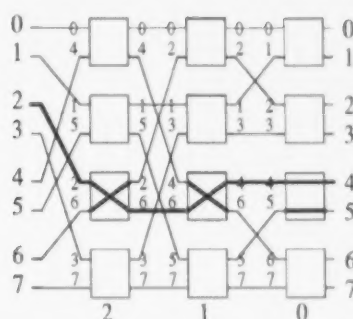


Figure 2.11: A size 8 GCN serving a request from input 2 to output 4.

More properties of the GCN can be found in [112, Ch. 5]. Note that if the switches labeled (1,3) and (4,6) in stage 1 are swapped without disconnecting any links, the Ω network is obtained (Figure 2.8 and 2.10). This immediately proves that for $N = 8$, the GCN and the Ω network are functionally equivalent (§ 2.5). Interestingly, they are derived in quite different ways. Siegel introduced fault tolerance to the GCN by adding an extra stage to the input side [112, Ch. 5]. The Extra Stage Cube (ESC) network in Figure 2.12 is obtained. Additional hardware is needed in order to select one of the 2 alternative paths between any input/output pair. Although the ESC is still blocking, most single faults can be tolerated.

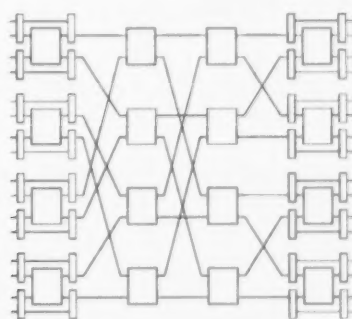


Figure 2.12: An ESC network of size 8.

2.4.4 Beneš' Rearrangeable Network

Beneš derives this important rearrangeable network from a rearrangeable $v(n, m, r)$ Clos network [11, 12]. He shows that the use of small switches, such as the 2×2 crossbar, can reduce the cost. Large crossbar switches in a size $N = nr = 2^k$ (for some integer k) rearrangeable Clos network should be recursively replaced by rearrangeable Clos networks of smaller sizes (Figure 2.13a). The resulting network has $2 \lg N - 1$ stages and $N \lg N - \frac{N}{2}$ switches (Figure 2.13b). Note that its left-most (right-most) $\lg N$ stages form a baseline (inverse baseline) network.

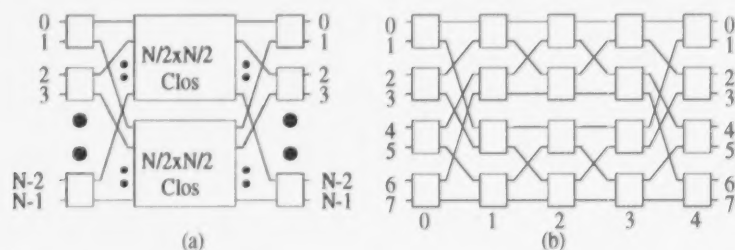


Figure 2.13: Beneš' rearrangeable network (b) and its recursive model (a).

There are $\frac{N}{2}$ alternative paths from any input to any output. The optimal sequential routing algorithm, the *looping algorithm* [41], takes $O(N \log N)$ time. Permutation routing on Beneš' network is closely related to 2-coloring problems on bipartite multi-graphs (§ 2.4.1). There are several generalizations of Beneš' rearrangeable network in the literature (e.g., [87, 132]).

2.4.5 The Data Manipulator Network

The Data Manipulator (DM) network is proposed by Feng [40] as a multi-stage version of the static network plus-minus-2- i (PM2I). A PM2I network is a chordal ring of N nodes $\{0, 1, \dots, N-1\}$, in which chords connect each node j to nodes $(j \pm 2^i) \bmod N$, for $0 \leq i < \lg N$ (Figure 2.14a). All networks in the class of DM networks, including the Augmented Data Manipulator (ADM), the inverse ADM (IADM), and the Gamma network, are topologically equivalent (§ 2.5).

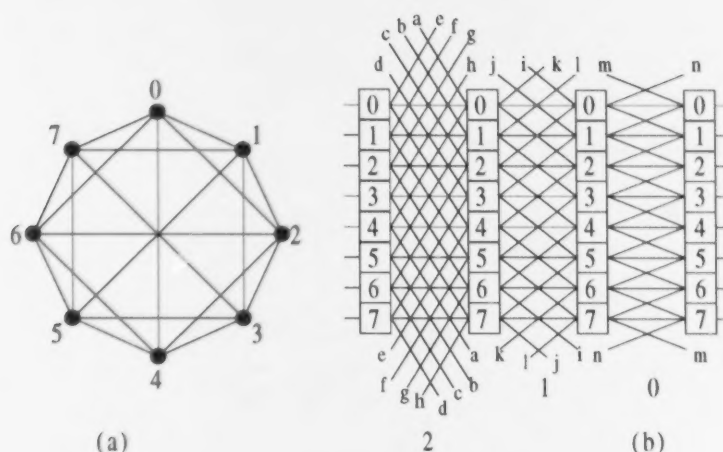


Figure 2.14: An 8-node PM2I network (a) and the corresponding DM network (b).

There are $\lg N$ stages of links and $\lg N + 1$ stages of switches in the DM network (Figure 2.14b). Stages of links/switches are numbered from right to left starting from 0. Switches in each stage are numbered from 0 downwards. Switch i in stage $s+1$ is only connected to switches numbered $(i - 2^s) \bmod N$, i , and $(i + 2^s) \bmod N$ in stage s , where $0 \leq i < N$. Every switch in the network, including 1×3 , 3×3 , and 3×1 switches, allows at most 1 connection at any time. Furthermore, several switches share one common control line. The DM can perform many “data manipulation” functions, including all GCN-admissible permutations [40].

The ADM network proposed by Siegel [112, Ch. 5] allows every switch to be controlled independently. Efficient algorithms for routing 1-to-1 and broadcast requests can be found in [112, Ch. 5]. Although the number of distinct ADM-admissible permutations is known [76, 77], the set of admissible permutations has not been characterized and no polynomial-time routing algorithm was available.

The IADM network studied by McMillen and Siegel [85] is an ADM network traversed in reverse. The Gamma network presented by Parker and Raghavendra [94] is an IADM in which every 3×3 switch allows 3 connections to coexist. Regarding combinatorial power, we have: $\text{Gamma} > \text{ADM} = \text{IADM} > \text{DM}$. The performance of DM type networks operating in packet-switching mode is studied in [133].

2.5 Equivalence of Networks

There are two kinds of equivalence relations we want to distinguish [124, Ch. 4]:

Topological equivalence: The corresponding graphs (i.e., switches modeled as nodes and links modeled as edges) of two networks are isomorphic.

Functional equivalence: Two networks are able to perform exactly the same set of permutations, after relabeling the inputs and outputs of a network if necessary.

More formal treatments of equivalence problems of networks appear in [16, 17, 67, 136]. Most studies of equivalence relations among networks concentrate on the class of $\log N$ -stage *banyan* networks. Using the GCN as the benchmark, Siegel and Smith [115] show that the Flip, Omega, GCN, and Indirect Binary n -cube networks are topologically equivalent. Wu and Feng [127] introduce the baseline and prove that the Flip, Omega, Indirect Binary n -cube, SW-banyan ($S=F=2$), and inverse baseline are all topologically equivalent. Pradhan and Kodandapani [101] define an equivalence relation and show that the Flip, Omega, Indirect Binary n -cube, SW-banyan, and all of their inverses are equivalent under the defined relationship. Thus, most of the proposed $\log N$ -stage banyan networks are topologically equivalent. However, they are in general functionally non-equivalent because of the differences in their admissible permutations, switch architectures, and control schemes (e.g., all DM type networks are topologically equivalent but they admit different sets of permutations).

Agrawal [3] initially studied the necessary and sufficient condition for topological equivalence among $\log N$ -stage banyan networks. Bermond and Fourneau gave a concise solution in [15]. Note that, by definition, every rearrangeable, wide-sense nonblocking, and strict-sense nonblocking network is functionally equivalent to any other network in the same class. Hardware cost and routing complexity (§ 2.6) become the primary concerns when selecting networks within such classes.

2.6 Figures of Merits

Like many other engineering exercises, the design of MINs typically involves trade-offs between conflicting factors. This section presents a collection of typical areas and questions of interest for evaluating MINs. More details can be found in [41, 82].

Capability: Is the network blocking, rearrangeable, or nonblocking? What is the set of admissible permutations? How many passes through the network are needed to route a given packet? What is the stage latency (stage-to-stage delay)? Is the bandwidth high enough to handle full traffic load and avoid congestion? How many transfers can proceed in parallel? Can the network be pipelined? (Layered MINs are relatively easy to pipeline.) Is it easy to simulate other networks?

Functionality: Can broadcasting be supported easily? Can the network combine packets targeting for the same destination to alleviate hot-spot problems? Is the routing algorithm deadlock- and starvation-free [33]? Can the network be partitioned into independent sub-networks?

Reliability: Is the network fault tolerant? How many faults can be tolerated? Can graceful degradation be achieved and how much disruption will it cause?

Cost: Is the number of switches (or links) required acceptable? Is the topology area-efficient when implemented using VLSI technology? (e.g., *cellular* INs [62] are easier to fabricate.) Is the routing algorithm easy to implement and fast?

Flexibility: Can the network adapt to various traffic loads and patterns? Is it possible to grow the network incrementally? Is the network reconfigurable?

Regularity: Does the network possess a symmetric structure that can be exploited to solve routing problems? Can the regularity in the topology simplify fabrication?

2.7 Complexity of Permutation Routing

Before we close this chapter, it is instructive to examine the complexity of permutation routing in MINs. This problem has practical significance because one way of maximizing the throughput of a clocked packet-switching network is to route each set of traffic requests that constitute a permutation (between the inputs and outputs) in 1 pass through the network, whenever possible. A parallel argument holds for MINs operating in circuit-switching mode where the traffic pattern is a permutation.

An efficient algorithm for deciding whether a given permutation π is admissible by a network G would be invaluable to the design of an efficient routing algorithm for that network. A direct solution to the above decision problem, however, does not seem to exist explicitly in the literature. We argue in the following theorem that it is unlikely that the above problem can be solved efficiently for any arbitrary MIN G . The 3-SAT problem [44], where each variable appears in at most 5 clauses, will be reduced to the above permutation routing problem. The reduction uses a special type of 2x2 switch that allows exactly 1 connection to be made (thus, leaving one output idle). A special switch can be constructed by cascading 2 conventional 2x2 switches (Figure 2.15).

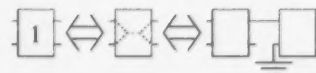


Figure 2.15: Special 2x2 switch allowing only one connection.

Theorem 2.5 *The problem of deciding whether a given permutation π is admissible by an arbitrary MIN G made of 2x2 (ordinary and special) switches is NP-complete.*

Proof: Given an instance of the 3-SAT problem, we construct a corresponding MIN G such that G admits the identity permutation if and only if the given boolean formula is satisfiable. For each variable x_i , $1 \leq i \leq n$, we have an input x_i^{in} and an output x_i^{out} . Connect 2 rows of 5 special switches between x_i^{in} and x_i^{out} such that 2 disjoint paths exist between x_i^{in} and x_i^{out} , corresponding to the true and false values of the variable x_i (Figure 2.16a). Call this part the x -network.

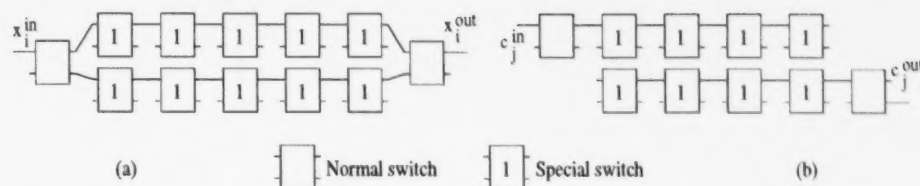


Figure 2.16: Constructing the MIN for (a) the variables, (b) the clauses.

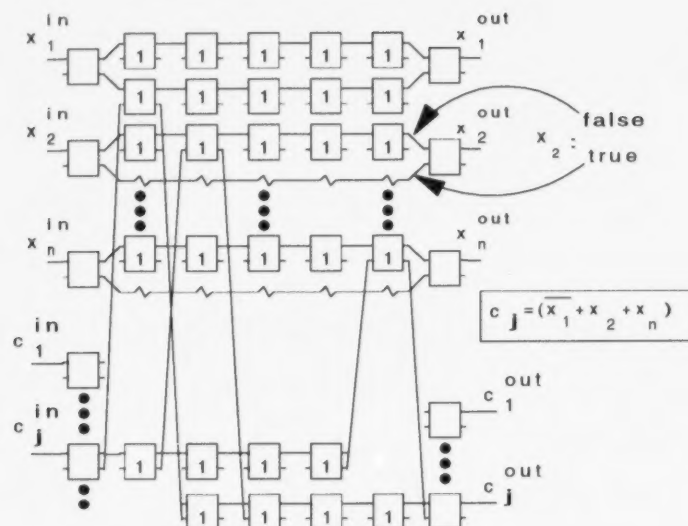
For each clause c_j in the given formula, $1 \leq j \leq m$, connect a thread of 4 special and 1 normal switches to input c_j^{in} , and another thread of 4 special and 1 normal switches to c_j^{out} . These 2 threads are separated (Figure 2.16b). Call this part the c -network. In both the x - and c -network all switches are numbered from left to right.

Since each variable appears at most 5 times, 5 iterations are needed to finish the connections. First, construct an undirected bipartite graph Γ with variables $x_1, x_2, \dots, x_n$ as nodes on the left side and clauses $c_1, c_2, \dots, c_m$ as nodes on the right side. Insert an edge between x_i and c_j for each occurrence of variable x_i in clause c_j . By coloring Γ using 5 colors (Vizing's theorem [21, p. 98]), we can decompose it into 5 connected components, say $\gamma_1, \gamma_2, \dots, \gamma_5$. Each connected component consists of edges bearing the same color number and all the nodes in Γ . In the s th iteration, we process γ_s . If there is an edge between x_i and c_j in γ_s , connect the output of the s th switch of the thread associated with c_j^{in} to the input of the $(s+1)$ -th switch in the upper (lower) row of x_i^{in} if the occurrence is x_i ($\bar{x}_i$). Also connect the output of this switch back to the input of the s th switch of the thread associated with c_j^{out} .

In the example in Figure 2.17, the literals $\bar{x}_1$, x_2 , and x_n in c_j are the 1st, 2nd and 5th occurrence of the variables x_1 , x_2 , and x_n , respectively. Finally, connect all the unused inputs and outputs to the ground. The MIN is now complete.

Suppose that the given formula $F = \bigwedge_{j=1}^m c_j$ has a satisfying assignment, then it is easy to see that G admits the identity permutation. This is true since satisfying a general clause c_j can be mapped to a path from c_j^{in} to c_j^{out} .

For the other direction suppose that G admits the identity permutation, then F is satisfiable because a truth assignment is directly readable from the routes. For

Figure 2.17: An example for $c_j = \overline{x_1} + x_2 + x_n$.

example, if the path between x_2^{in} and x_2^{out} passes through the upper (lower) thread associated with variable x_2 , then x_2 is false (true). This completes the proof. $\square$

Corollary 2.1 *The above decision problem is NP-complete even if every switch is a 2×2 crossbar and the network has depth 12.*

Proof: If every special switch is replaced by an equivalent pair of normal switches (Figure 2.15), then we get a MIN made of normal 2×2 switches. Clearly, the resulting MIN has depth 12. $\square$

It is then natural to ask: what is the maximum depth d such that the above decision problem can be solved efficiently on any arbitrary MIN of depth $\leq d$? This problem seems to be open. In Theorem 2.6 we give a direct solution for $d = 2$. Appendix A proves that $O(N)$ time is enough for $d = 3$.

Theorem 2.6 *For a 2-stage MIN of size N , made of 2×2 switches, it takes $O(N)$ time to determine whether a given permutation can be satisfied in 1 pass.*

Proof: Denote the switch that is directly connected to input/output t by $s(t)$. Assume all switches are initially set to a state called *unknown* (U). Given a permutation

π , it takes $O(1)$ time to identify any of the following 3 exclusive cases for each 1-to-1 request (i, j) in π :

1. $s(i)$ and $s(j)$ are not connected by any link: so π cannot be realized in 1 pass.
2. One link connects $s(i)$ to $s(j)$: if both $s(i)$ and $s(j)$ are in the U state then set them to straight or swap as required. Otherwise, if the required setting conflicts with the current state(s), then terminate with failure. If there is no conflict, set the switch in state U to the required state and proceed to the next request.
3. Two links connect $s(i)$ to $s(j)$: set $s(i)$ and $s(j)$ as required if they are in the unknown state. Otherwise, proceed to the next request.

Hence, the routing time and storage requirement are both $O(N)$. $\square$

In light of the above discussion, it is interesting to settle the complexity of routing permutations on arbitrary MINs of depth d where $4 \leq d \leq 11$.

Chapter 3

Routing on Extra Stage Networks

In this chapter we introduce the class of k -extra-stage Generalized Cube Networks (k -GCNs) obtained by adding k extra stages to the class of Generalized Cube Networks (GCNs). Networks of this type provide 2^k alternative paths between each input/output pair. Given such a multi-path MIN and a prescribed traffic pattern, we ask what path-selection strategy minimizes the network delay. § 3.1 gives an introduction to this research area. Switch model and notation are covered in § 3.2. Useful symmetry properties of the k -GCN are derived in § 3.3. § 3.4 proves that under some assumptions, distributing the packets evenly over the 2 alternative paths minimizes the average delay in a 1-GCN for all traffic patterns. The simulation experiment presented in § 3.5 verifies our argument. § 3.6 gives some concluding remarks.¹

3.1 Introduction

Multi-path MINs provide more than one path for some input/output pairs. For example, the ESC network (Figure 2.12) provides 2 alternative paths between any input/output pair. The C2SC network [1] has 2 extra stages added to a GCN (§ 2.4.3) so 4 alternative paths are available. Under the control of fault tolerant routing algo-

¹A preliminary version of this chapter was presented at the Seventh Annual Canadian High Performance Computing Symposium in Calgary, Canada [71].

rithms, such multi-path MINs can tolerate single or multiple switch or link failures depending on their capabilities.

When there is no fault in a multi-path MIN operating in packet-switching mode, the flexibility in path-selection offers an opportunity for improving the network delay, which is the average number of *clock cycles* (assume a global clock synchronizes the transfer of packets between stages) required for a packet to pass through the network. The price for taking advantage of the alternative paths is that packets resulting from a long message may not arrive at their destination in order. A procedure called *resequencing* [58] is needed. By using a suitable path-selection strategy, the traffic load in the network can be balanced to alleviate congestion. Intuitively, an optimal path-selection strategy depends on the network topology as well as the traffic pattern. In this chapter, we focus on the following class of recursive networks obtained by adding 1 or more extra stages to GCNs:

Definition 3.1 A 1-GCN is obtained by connecting a column of $\frac{N}{2}$ 2×2 switches to the input side of a size N GCN. A k -GCN, where $2 \leq k < \lg N$, is constructed by using a perfect shuffle permutation [117] to connect a column of $\frac{N}{2}$ 2×2 switches to the left side of a $(k-1)$ -GCN of size N .

Figure 3.1 sketches the basic idea for $k \leq 2$. In general, any k -GCN provides 2^k link-disjoint paths between every input/output pair of terminals. Interest in the class

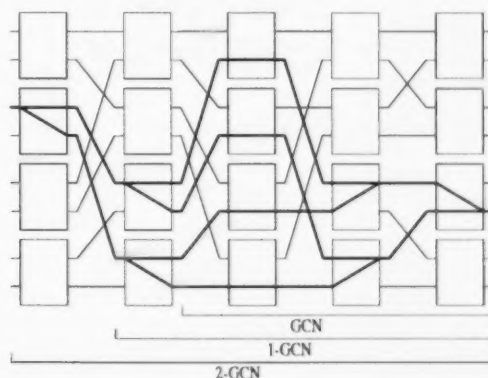


Figure 3.1: 1-GCN and 2-GCN of size 8.

of k -GCN networks arises from the fact that the 1- and 2-GCNs are topologically equivalent to the ESC and C2SC networks, respectively. Moreover, adding extra stages to the GCN provides an *upgrade path* from blocking to rearrangeable networks. Note that using more than $\lg N - 1$ stages does not increase the number of link-disjoint paths between any input/output pair.

Our path-selection problem can now be stated as a routing problem, where the traffic pattern is modeled by an $N \times N$ matrix L where l_{ij} in L is the probability of having a packet generated by input i , which is targeted to output j , during any clock cycle. Packet generation at inputs are independent Poisson processes. Given a k -GCN and a traffic matrix L , what packet routing strategy minimizes the network delay? In this chapter we focus on the 1-GCN. A path-selection scheme is described by *usage probabilities* (explained later) assigned to the alternative paths.

In a k -GCN, the k extra stages and the original GCN can be viewed as 2 sub-networks, and routing can be broken down into 2 steps: a *path-selection* step done at the extra stages, and a routing step done in the GCN sub-network. Once a sub-path has been chosen at the extra stages, the remaining part of the path through the GCN sub-network is readily computable (two efficient routing tag schemes are given in § 2.4.3). Therefore, we focus on the path-selection step.

Before we proceed, some related work is reviewed. Recently, Elmallah and Culberson [34] studied the multi-commodity flow problem on another generalization of 1-GCN, in connection with circuit-switching routing. Their results lead to efficient algorithm for determining whether a given permutation is admissible in a 1-GCN where all flow demands and capacities are 0/1-valued. The performance of some circuit-switching MINs is studied in [130] using Markov chain and simulation.

Analytical and simulation models for estimating some important performance measures, such as network throughput (bandwidth) and end-to-end delays, for packet-switching unbuffered and buffered networks exist in the literature. Most studies focus on cube-type networks. To make the analysis tractable, it is often assumed that input packets are *independently* directed to each network output. Classical results in this

area [31, 65, 96, 100] add the restriction that requests from a given input to all outputs are equi-probable (using our notation, $l_{ij} = \frac{1}{N}$ everywhere). More recent results relax this restriction for banyan networks.

Not as many results exist for choosing an optimal routing strategy for multi-path networks under the general, or even the equi-probable, traffic pattern. Jean-Marie recently [58] show that by assigning a $\frac{1}{2^m}$ usage probability to each of the 2^m alternative paths in an indirect binary n -cube network (which is a GCN traversed in reverse) with m extra stages, the network delay is minimized. We compare our approach with that in [58] at the end of this chapter.

3.2 Switch Model and Notation

To analyze our routing problem, assume every switch has 2 first-in-first-out queues of unlimited capacity (Figure 3.2). At the beginning of every cycle, if there is a packet at one of the queue heads, it leaves the switch in that cycle. If there are 2 packets at the heads of the 2 queues, requiring different outputs, both of them proceed to the next stage in the same cycle. If the 2 packets at the heads require the same output, then a *conflict* occurs. One of the packets advances to the next stage in the current cycle while the other is delayed to the next cycle. At the end of every cycle, 1 packet may arrive from each input of a switch and join the corresponding queue.

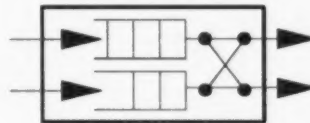


Figure 3.2: 2x2 switch with unlimited buffer queues.

Every switch maintains a *priority bit*, which is initialized randomly, for resolving conflicts. When a conflict occurs, the packet in the upper (lower) queue proceeds if the priority bit is 0 (1). The priority bit is toggled after every conflict such that if a conflict occurs at time t , the delayed packet leaves the switch at time $t+1$, even if

another conflict occurs. Thus, each packet is delayed by at most 1 clock cycle (since its arrival to the front of a queue) at each stage due to conflicts.

The binary representation of an n -bit number b is $(b_{n-1} \dots b_1 b_0)$. A 'c' in a bit string signifies a "don't care" bit. We adopt the following labeling scheme for the k -GCN. The $n + k$ stages are numbered from right to left starting from 0. The $\frac{N}{2}$ switches in each stage are numbered from top to bottom starting from 0. Switch inputs/outputs in every stage are also labeled from top to bottom starting from 0. An example is in Figure 3.3. Under the current labeling scheme, the cube function implemented by the connection pattern on the left side of stage s , where $s < n-1$, in a GCN becomes: $\text{cube}_s^*(b_{n-1} \dots b_1 b_0) = (b_{n-1} \dots b_{s+2} b_0 b_s \dots b_1 b_{s+1})$. At the k extra stages, a path is numbered $t = (t_{k-1} \dots t_1 t_0)$, where $t_i = 0$ (1) if the upper (lower) output of the switch at stage $i + n$ is used. For convenience, we include below a glossary of symbols used throughout the chapter.

c	average number of conflicts experienced by all packets during one cycle (that is, each conflict adds 2 to the sum).
c_{ij}	average number of conflicts seen by a packet from i to j , taking any path.
c_{ijk}	average number of conflicts seen by a packet from i to j , taking path k .
c_{ijks}	average number of conflicts seen by a packet from i to j at stage s , taking path k .
r_{ijk}	set of switches used by a packet from i to j , taking path k .
p_{ijk}	probability of using path k for a packet from i to j .
l_{ij}	probability of generating a packet from i to j in any cycle.
Φ_{ijst}	set of network outputs reachable from the switch in stage s in r_{ijst} .
Ψ_{ijst}	set of network inputs that can reach the switch in stage s in r_{ijst} .

3.3 Properties of the k -GCN

Some properties of the k -GCN are captured by the next 2 lemmas (the details in their proofs are not crucial to the understanding of our analysis in § 3.4.2). Lemma 3.2 uses Lemma 3.1, while the main Theorem 3.3 needs the symmetry property in Lemma 3.2. Let r_{ijst} be the sequence of switches in path t between input i and output j . Let $f_s(i, j, t)$ be the switch number at stage s used by a packet on route r_{ijst} .

Lemma 3.1 *In every stage $s = k, k+1, \dots, n-1$, a switch with number y is visited only by packets that are taking their path labeled $y \bmod 2^k$, where $0 \leq y < \frac{N}{2}$.*

Proof: Consider an imaginary packet from an arbitrary network input $i = (i_{n-1} \dots i_1 i_0)$ taking path $t = (t_{k-1} \dots t_1 t_0)$. After passing the k extra stages, this packet arrives at the switch input numbered $(i_{n-k-1} \dots i_1 t_{k-1} \dots t_0 i_{n-k})$ at stage $n-1$, where $(i_{n-k-1} \dots i_1)$ becomes a null string if $n-k-1 < 1$. The corresponding switch is $f_{n-1}(i, j, t) = (i_{n-k-1} \dots i_1 t_{k-1} \dots t_1 t_0)$. Since $s \geq k$, the bits $(b_k \dots b_1)$ are never affected by the cube_s^* function (page 36). These positions precisely correspond to the t bits in $(i_{n-k-1} \dots i_1 t_{k-1} \dots t_0 i_{n-k})$. So the set of switches reachable by the imaginary packet can be described by $(\underbrace{cc \dots c}_{n-k-1} t_{k-1} \dots t_0)$. Therefore, switches in stage s , where $k \leq s < n$, can be partitioned into 2^k classes. $\square$

The example in Figure 3.3 shows that switches in stages 2 and 3 can be partitioned into classes '00', '01', '10', and '11.' Lemma 3.2 reveals a useful symmetry property of the k -GCN. Recently, a similar property has been used to solve flow problems [34].

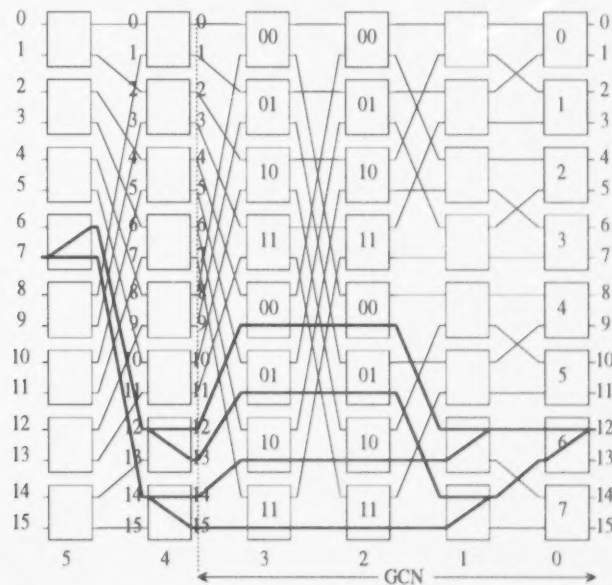


Figure 3.3: Partitioning switches in a 2-GCN of size 16 into 4 classes.

Lemma 3.2 *Let $i, j \in \{0, 1, \dots, N-1\}$ be a network input and a network output, respectively. Let $s \in \{0, 1, \dots, n+k-1\}$ be a stage number, and $t \in \{0, 1, \dots, 2^k-1\}$ be a path number. Let Ψ_{ijst} be the set of network inputs that can reach the switch at stage s on route r_{ijt} . Similarly, let Φ_{ijst} be the set of network outputs reachable from the switch at stage s on route r_{ijt} . Then, for any path numbers $\tau', \tau'' \in \{0, \dots, 2^k-1\}$:*

1. $\Psi_{ijs\tau'} = \Psi_{ijs\tau''}$, and
2. $\Phi_{ijs\tau'} = \Phi_{ijs\tau''}$.

Proof: We prove the two statements in sequence by conducting a case analysis.

1. The special case $s = n+k-1$ is obvious since all (i, j) routes pass through 1 switch, so $\Psi_{ij(n+k-1)\tau'} = \Psi_{ij(n+k-1)\tau''}$. The rest of the proof considers 3 cases. Examine the extra stages first. After passing stage q , $q \in \{n+k-1, \dots, n\}$, a packet from an arbitrary input $i = (i_{n-1} \dots i_1 i_0)$ taking path $\tau = (\tau_{k-1} \dots \tau_1 \tau_0)$ visits the input $(i_{q-k-1} \dots i_1 \tau_{k-1} \dots \tau_{q-n} i_{q-k})$ (Lemma 3.1) at stage $q-1$. The set of network inputs that have access to the corresponding switch $f_{q-1}(i, j, \tau) = (i_{q-k-1} \dots i_1 \tau_{k-1} \dots \tau_{q-n})$ can be described by $(\underbrace{cc \dots c}_{n+k-q} i_{q-k-1} \dots i_1 c)$. Since this bit string does not contain the τ bits, the same set of network inputs have access to the switch visited at stage q by the arbitrary packet no matter which path is taken. Hence, we have $\Psi_{ijs\tau'} = \Psi_{ijs\tau''}$ for $s \in \{n+k-1, \dots, n-1\}$.

We now consider $q \in \{n-1, \dots, k+1\}$ (skip this case if $n-1 < k+1$). This region covers the left part of the GCN sub-network. After passing stage q , the arbitrary packet heading to network output $j = (j_{n-1} \dots j_1 j_0)$ enters the switch input numbered $(j_{n-1} \dots j_q i_{q-k-1} \dots i_1 \tau_{k-1} \dots \tau_0 i_{q-k})$ at stage $q-1$. The corresponding switch $f_{q-1}(i, j, \tau) = (j_{n-1} \dots j_q i_{q-k-1} \dots i_1 \tau_{k-1} \dots \tau_0)$ is accessible from the set of network inputs described by $(\underbrace{cc \dots c}_{n-k-q} i_{q-k-1} \dots i_1 c)$. Again, the bits $\tau_{k-1} \dots \tau_0$ are absent from the bit pattern. Hence, $\Psi_{ijs\tau'} = \Psi_{ijs\tau''}$ for $s \in \{n-2, \dots, k\}$.

The third case $q \in \{k-1, \dots, 0\}$ is easy because every switch in this portion is accessible from all network inputs. It could be seen by putting $q = k$ into

$(\underbrace{cc \dots c}_{n-q+k} i_{q-k-1} \dots i_1 c)$ to get all don't care bits. We conclude that, in general, $\Psi_{ijs\tau'} = \Psi_{ijs\tau''}$ for all $s \in \{n+k-1, \dots, 0\}$.

2. We study 3 cases again. Consider stage $q \in \{n, \dots, k+1\}$, i.e., the left part of the GCN sub-network. Again, we trace an arbitrary packet coming from network input $i = (i_{n-1} \dots i_1 i_0)$, taking path $\tau = (\tau_{k-1} \dots \tau_1 \tau_0)$, heading to network output $j = (j_{n-1} \dots j_1 j_0)$. It enters the switch input numbered $(i_{n-k-1} \dots i_1 \tau_{k-1} \dots \tau_0 i_{n-k})$ at stage $n-1$. In general, after passing stage q , the packet enters the switch input numbered $(j_{n-1} \dots j_q i_{q-k-1} \dots i_1 \tau_{k-1} \dots \tau_0 i_{q-k})$ at stage $q-1$. The associated switch is $f_{q-1}(i, j, \tau) = (j_{n-1} \dots j_q i_{q-k-1} \dots i_1 \tau_{k-1} \dots \tau_0)$. The set of network outputs reachable from this switch can be described by $(j_{n-1} \dots j_q \underbrace{cc \dots c}_q)$, which is dependent on the destination address but not the path number τ . Thus, we have $\Phi_{ijs\tau'} = \Phi_{ijs\tau''}$ for $s \in \{n-1, \dots, k\}$.

The second case has $q \in \{k, \dots, 1\}$. The packet enters the switch input numbered $(j_{n-1} \dots j_q \tau_{q-2} \dots \tau_0 i_{q-1})$ at stage $q-1$ after passing stage q . The associated switch is $f_{q-1}(i, j, \tau) = (j_{n-1} \dots j_q \tau_{q-2} \dots \tau_0)$. This switch can reach network outputs described by $(j_{n-1} \dots j_q \underbrace{cc \dots c}_q)$. Applying the same argument as above, we have $\Phi_{ijs\tau'} = \Phi_{ijs\tau''}$ for $s \in \{k-1, \dots, 0\}$.

In the third case $s \in \{n+k-1, \dots, n\}$, clearly any switch in this portion can access all network outputs. Hence, we conclude that $\Phi_{ijs\tau'} = \Phi_{ijs\tau''}$ for $s \in \{n+k-1, \dots, 0\}$. $\square$

3.4 Load Balancing on the 1-GCN

In this section, we first introduce basic concepts and formulas for solving the path-selection problem on the 1-GCN. Then, the main theorem in this chapter is derived.

3.4.1 Basic Concepts and Definitions

Since every input i can generate at most 1 packet per cycle, we have $\sum_{j=0}^{N-1} l_{ij} \leq 1$ for all input i . Once an input i , an output j , and a path number k , where $k \in \{0, 1\}$, are given, the sequence of switches in the path r_{ijk} can be uniquely determined. Let p_{ijk} be the probability of assigning path k to a packet going from i to j . Clearly:

$$p_{ij1} = 1 - p_{ij0} \quad (3.1)$$

for a 1-GCN. Thus, there is only one independent variable (p_{ij0}) for every input/output pair. When a packet going from i to j arrives at stage n , the router generates a 0/1 random integer such that the probability of getting a 0 is p_{ij0} . The random number represents the path assignment.

In steady state, minimizing the average network delay is equivalent to minimizing the average queue length. Every conflict increments the queue length by 1. Hence, we focus on the minimization of conflicts. Let c be the average of the total number of conflicts experienced by all the packets generated in the network in one cycle. Then,

$$c = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} c_{ij} \cdot l_{ij}, \quad (3.2)$$

where c_{ij} is the expected number of conflicts experienced by a packet going from input i to output j . c_{ij} can be written as:

$$c_{ij} = \sum_{k=0}^1 p_{ijk} \cdot c_{ijk}, \quad (3.3)$$

where c_{ijk} is the expected number of conflicts experienced by a packet going from i to j if path k is chosen. Finally, we have:

$$c_{ijk} = \sum_{s=0}^n c_{ijk s}, \quad (3.4)$$

where $c_{ijk s}$ is the expected number of conflicts experienced at stage s by a packet going from i to j , taking path k .

3.4.2 Probability Assignment

The main result in this chapter is presented in the following 2 theorems:

Theorem 3.3 *In steady state, setting $p_{ij0} = 0.5$ for all input i and output j gives a stationary point for c (i.e., $\frac{\partial c}{\partial p_{\alpha\beta 0}}$ vanishes at $p_{ij0} = 0.5$), regardless of the traffic pattern.*

Proof: Recall that c_{ijks} is the expected number of conflicts experienced at stage s by a packet from i to j taking path k . In steady state, we have:

$$c_{ijks} = \sum_{m \in \Psi'_{ijsk}} \sum_{n \in \Phi'_{ijsk}} l_{mn} p_{mnk}, \quad (3.5)$$

where $\Psi'_{ijsk} = \Psi_{ijsk} - \Psi_{ij(s+1)k}$, for $n \geq s \geq 1$,

and $\Phi'_{ijsk} = \Phi_{ij(s-1)k}$, for $n \geq s \geq 0$.

Figure 3.4 depicts how a conflict occurs. For the boundary cases, we define $\Psi'_{ij0k} = \{0, 1, \dots, N-1\}$, $\Psi_{ij(n+1)k} = \{i\}$ and $\Phi_{ij(-1)k} = \{j\}$. Following [58], we ignore stage n , which is the path-selection stage (as explained in [58], the “Bernoulli” switches in stage n are fast because there is no need to examine the packet headers and no computation is done, so the service time at these switches is small with respect to the inter-arrival time. Consequently, packets do not interfere with each other in stage n , generally speaking.). We also ignore stage 0 because conflicting packets are targeted to the same output terminal; and this “serialization” of packets is always needed, regardless of p_{ij0} . (In fact, the delay of packets at stage 0 is counted towards the cost of resequencing, which is studied in [58].) The expected number of conflicts experienced by a packet going from i to j taking path k is:

$$c_{ijk} = \sum_{s=1}^{n-1} c_{ijks} = \sum_{s=1}^{n-1} \sum_{m \in \Psi'_{ijsk}} \sum_{n \in \Phi'_{ijsk}} l_{mn} p_{mnk}. \quad (3.6)$$

The expected number of conflicts experienced by a packet going from i to j , taking either path, is:

$$c_{ij} = \sum_{k=0}^1 p_{ijk} \sum_{s=1}^{n-1} c_{ijks} = \sum_{k=0}^1 \sum_{s=1}^{n-1} p_{ijk} c_{ijks}$$

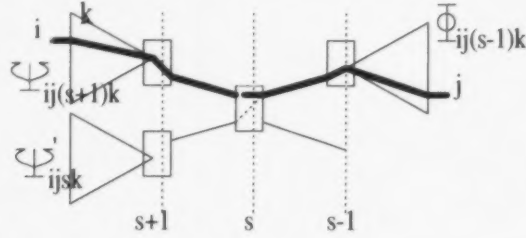


Figure 3.4: The scenario in which a conflict may occur.

$$= \sum_{s=1}^{n-1} c_{ij0s} p_{ij0} + \sum_{s=1}^{n-1} c_{ij1s} (1 - p_{ij0}).$$

Equation 3.1 was applied in the last step. Now we apply Equation 3.6 to get:

$$\begin{aligned} c_{ij} &= \sum_{s=1}^{n-1} p_{ij0} \sum_{m \in \Psi'_{ijs0}} \sum_{n \in \Phi'_{ijs0}} l_{mn} p_{mn0} + \sum_{s=1}^{n-1} (1 - p_{ij0}) \sum_{m \in \Psi'_{ijs1}} \sum_{n \in \Phi'_{ijs1}} l_{mn} (1 - p_{mn0}) \\ &= \sum_{s=1}^{n-1} \left\{ p_{ij0} \left[\sum_{m \in \Psi'_{ijs0}} \sum_{n \in \Phi'_{ijs0}} l_{mn} p_{mn0} - \sum_{m \in \Psi'_{ijs1}} \sum_{n \in \Phi'_{ijs1}} l_{mn} (1 - p_{mn0}) \right] + \right. \\ &\quad \left. \sum_{m \in \Psi'_{ijs1}} \sum_{n \in \Phi'_{ijs1}} l_{mn} (1 - p_{mn0}) \right\}. \end{aligned}$$

Lemma 3.2 allows us to combine the summations inside the square bracket to yield:

$$c_{ij} = \sum_{s=1}^{n-1} \{ p_{ij0} [\sum_{m \in \Psi'_{ijs0}} \sum_{n \in \Phi'_{ijs0}} l_{mn} (2p_{mn0} - 1)] + \sum_{m \in \Psi'_{ijs0}} \sum_{n \in \Phi'_{ijs0}} l_{mn} (1 - p_{mn0}) \}. \quad (3.7)$$

The average number of conflicts experienced by all the packets in one cycle is given by Equation 3.2. To minimize the average number of conflicts we first solve the following equation:

$$\frac{\partial c}{\partial p_{\alpha\beta 0}} = 0,$$

where $0 \leq \alpha < N$ and $0 \leq \beta < N$. Substituting Equation 3.2 for c gives:

$$\sum_i \sum_j \frac{\partial c_{ij}}{\partial p_{\alpha\beta 0}} l_{ij} = 0.$$

Using Equation 3.7, we obtained:

$$\sum_i \sum_j \sum_s \{ [\sum_m \sum_n l_{mn} (2p_{mn0} - 1)] \frac{\partial p_{ij0}}{\partial p_{\alpha\beta 0}} + 2p_{ij0} \sum_m \sum_n l_{mn} \frac{\partial p_{mn0}}{\partial p_{\alpha\beta 0}} \}$$

$$\begin{aligned}
& - \sum_m \sum_n l_{mn} \frac{\partial p_{mn0}}{\partial p_{\alpha\beta 0}} \} l_{ij} = 0 \\
& \sum_i \sum_j \sum_s \{ \sum_m \sum_n l_{mn} (2p_{mn0} - 1) \frac{\partial p_{ij0}}{\partial p_{\alpha\beta 0}} + (2p_{ij0} - 1) \sum_m \sum_n l_{mn} \frac{\partial p_{mn0}}{\partial p_{\alpha\beta 0}} \} l_{ij} = 0, \quad (3.8)
\end{aligned}$$

We now see that $p_{xy0} = 0.5$ for all x 's and y 's is a solution to the above system of linear homogeneous equations. The same procedure can be repeated for all α 's and β 's. $\square$

The nature of the point $p_{ij0} = 0$ generally depends on the l_{ij} 's. Note that the method of Lagrange multipliers is a standard procedure for determining whether a multi-variable minimization problem has a unique solution (see, for example, [122, Ch. 4], [25, Ch. 8], and [28, Ch. 7]). However, this method is too complex except for a small and fixed number of variables. So we focus on a special case, in which one global p_0 is used for all inputs and outputs (using one global p_0 is reasonable because it reduces software/hardware overhead), such that the nature of the solution $p_0 = 0.5$ can be readily determined.

Theorem 3.4 *If $p_{ij0} = p_0$ for all i 's and j 's, then c reaches a unique minimum when $p_0 = 0.5$, regardless of the traffic pattern.*

Proof: It is easy to see that Equation 3.7 reduces to:

$$\begin{aligned}
c_{ij} &= \sum_{s=1}^{n-1} \{ (2p_0^2 - p_0) \sum_{m \in \Psi'_{ijs0}} \sum_{n \in \Phi'_{ijs0}} l_{mn} + (1 - p_0) \sum_{m \in \Psi'_{ijs0}} \sum_{n \in \Phi'_{ijs0}} l_{mn} \} \\
&= (2p_0^2 - 2p_0 + 1) \sum_{s=1}^{n-1} \sum_{m \in \Psi'_{ijs0}} \sum_{n \in \Phi'_{ijs0}} l_{mn}
\end{aligned}$$

Therefore, Equation 3.2 becomes $c = (2p_0^2 - 2p_0 + 1)\mathcal{K}$, where $\mathcal{K}$ is a positive constant². It is straight forward to prove that $p_0 = 0.5$ is a unique minimum because it is a quadratic equation. $\square$

² $\mathcal{K} = 0$ happens when the traffic pattern is so good that the packets never conflict with each other. This case is uninteresting but the solution $p_0 = 0.5$ remains valid.

3.5 Simulation Study

To verify the practicality of Theorem 3.4 and to observe transient behavior of the system, we conducted a simulation study. In subsequent sub-sections we present the experiment design, the results, the observations, and the interpretations.

3.5.1 Experiment and Simulator Design

The traffic pattern is either *uniform* (any input i has equal probability to send a packet to any output j) or it has *hotspots* (correspond to frequently referenced memory modules in real systems [98]). Hotspots are modeled by randomly selecting a hotspot output at the beginning of a simulation run. Any input i has $\frac{2}{N}$ probability to send a packet to the hotspot if it chooses to generate a packet in a cycle. Other outputs are equally likely to be visited.

The quantity to be observed is the average queue length in the 1-GCN. The primary independent variables are the p_{ij0} 's. To cut the number of independent variables, we set all $p_{ij0} = p_0 \in \{0, 0.2, 0.4, 0.5, 0.6, 0.8, 1.0\}$. The intensity of the workload is modeled by a parameter $w \in \{0.2, 0.3, 0.4, 0.5, 0.6\}$, which is the probability of generating a packet in a cycle by each input. Two network sizes, $N = 8$ and 16, are considered. Every configuration is simulated 4 times and the average response is calculated. The total number of runs is $7 \times 5 \times 2 \times 2 \times 4 = 560$. Each run lasts 5,000 cycles. Pilot runs indicate that transient behavior is negligible at the end of this period.

The simulator is written in C. It has been compiled for MS-DOS and UNIX machines. Two execution modes are supported.

1. The *interactive mode* allows the user to monitor all relevant quantities such as queue lengths at all switches, average queue length in every stage, average queue length in the network, routing tag of each packet, and the number of packets generated (received) for each input (output). Transient behavior can be observed easily (Figure 3.9). Cycle-by-cycle execution is also supported.

2. The *batch mode* allows fast execution by disabling screen output. Execution terminates after a prescribed number of cycles, with all statistics saved in a file.

3.5.2 Results and Interpretations

Eight series of graphs are plotted and grouped into 4 figures. Data and confidence interval (of the overall average queue length) are given in Appendix B. Note that in some configurations, the combined effect of p_0 and w causes the queue lengths at certain stages to grow indefinitely, i.e., the system never reaches steady state. Consequently, some data points are missing from the graphs.

It is worthwhile to examine the throughput limit of a single 2x2 switch before we proceed. Assume unlimited buffer size, evenly distributed traffic and 1 packet arrives from each input per cycle. There are 2 packets at the heads of the buffer queues trying to leave the switch at the end of every cycle. Since the probability of a conflict is 0.5, the best average throughput of a switch is $2 \times 0.5 + 1 \times 0.5 = 1.5$ packets per cycle, or 0.75 packets per link per cycle.

Below we analyze the graphs qualitatively. In all graphs we plot the queue length against p_0 . Except a few special cases and fluctuations, the average queue length of the network is easily seen to be minimal when $p_0 = 0.5$. In the graphs, w and p_0 are multiplied by 10 to eliminate the decimal point for better legibility.

1. *The effect of workload on queue lengths for $N = 8$ (Figure 3.5):*

Uniform traffic: When $w = 2$, the workload is too light to create appreciable congestion so the queue lengths fluctuate at a low level. When w goes from 3 to 6, it becomes increasingly evident that $p_0 = 0.5$ minimizes the average queue lengths. When w is high, even small deviations of p_0 from 0.5 "saturates" the network, i.e., the queue lengths grow indefinitely. For instance, when $w = 5$ and $p_0 = 0.2$ (0.8), switch 2 (switch 0) in stage 2 receives the traffic load $0.5 \times 2 \times 0.8 = 0.8 > 0.75$, so saturation is inevitable. Note that the queue length at stage 2 is consistently the longest. The path-

selection done in stage 3 causes congestion in stage 2 when p_0 deviates from 0.5. Stages 0 and 1 are subject to lower packet arrival rates.

Hotspot traffic: When $w = 2$, hotspot congestion occurs in stage 1 when $p_0 = 0$ or 1, and longer queues (c.f. uniform traffic) emerge. The trend continues to other graphs in the series. When $w \geq 4$, the queue length at stage 1 flattens when $p_0 = 0$ and 1 because the maximum throughput of the 2x2 switch limits the traffic flow from stage 2 into stage 1.

2. *Queue lengths at various stages for $N = 8$ (Figure 3.6):*

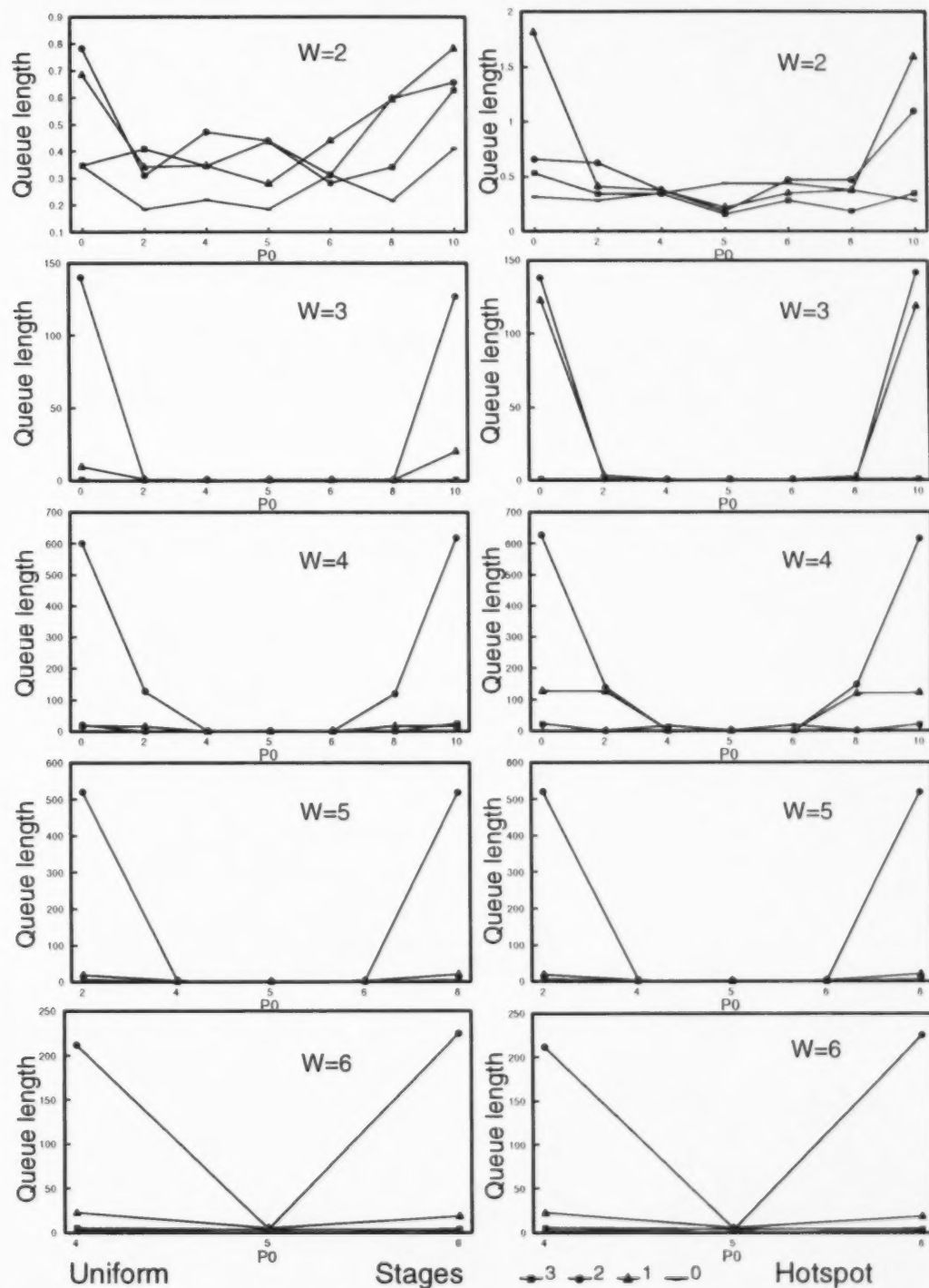
Stage 3: The curves for uniform and hotspot traffic are similar because the hotspot effect cannot be seen in this path-selection stage.

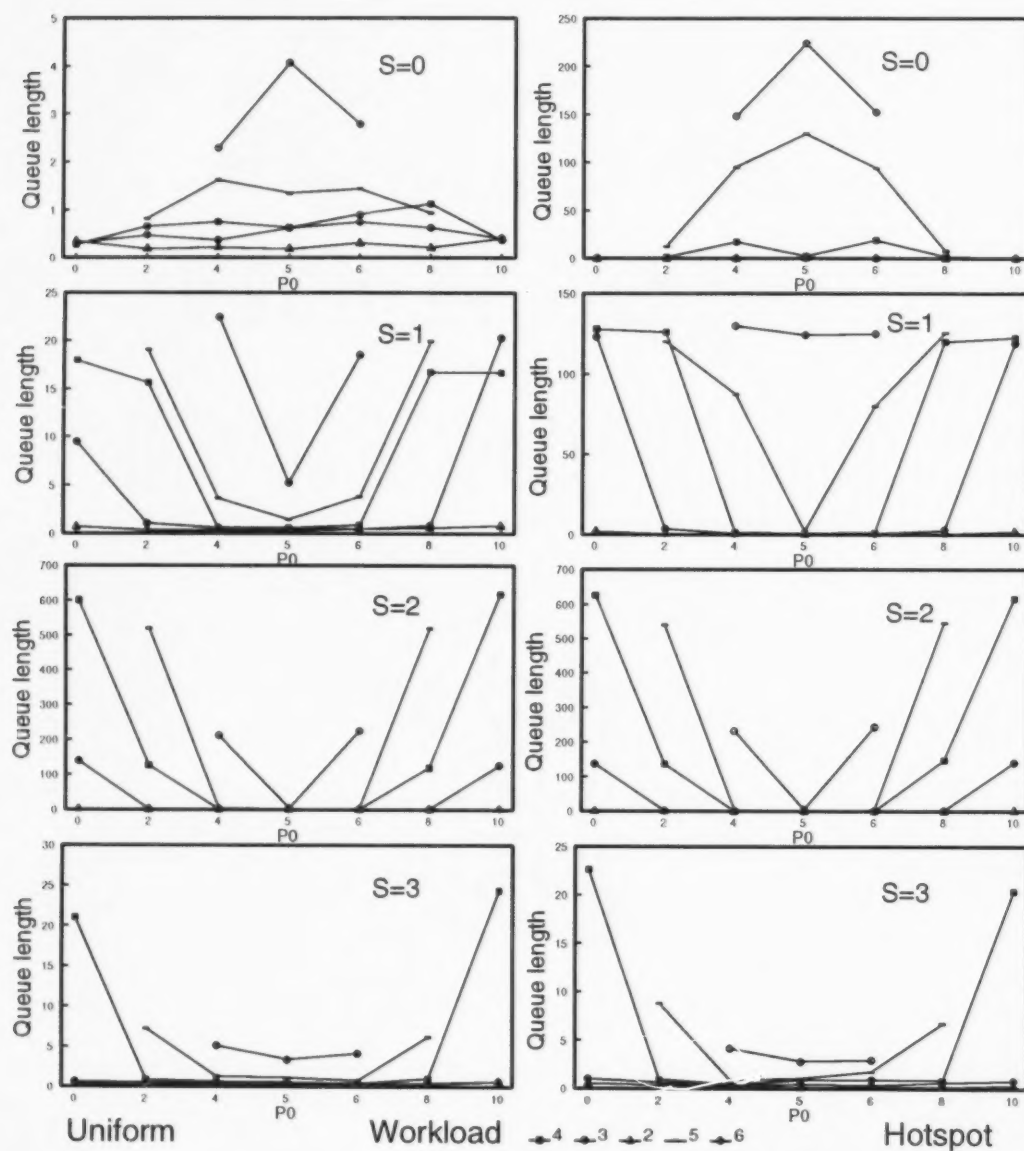
Stage 2: Hotspot traffic begins to give slightly longer queues.

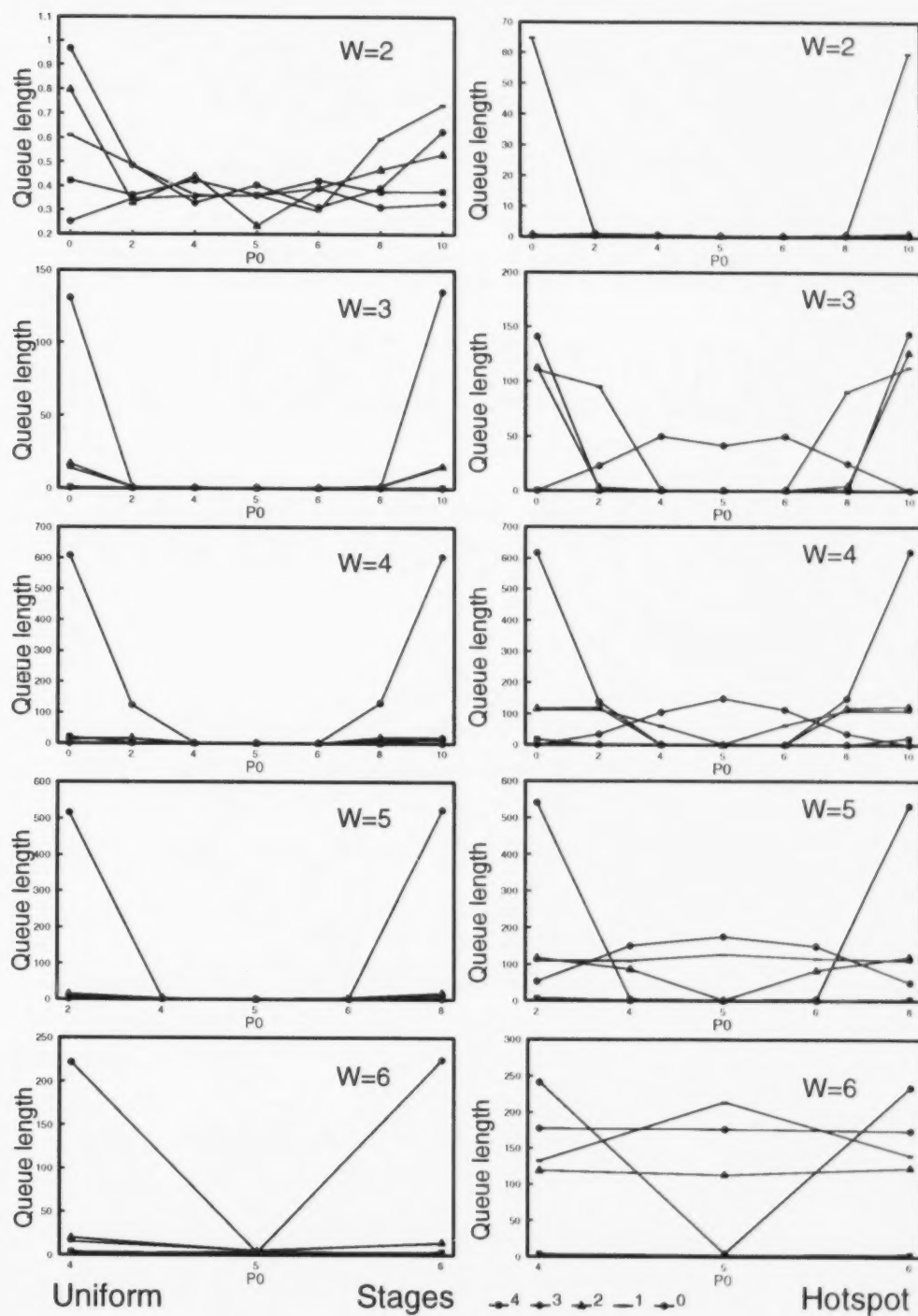
Stage 1: In both the uniform and hotspot graph, the curves are still v-shaped but flattening occurs when p_0 approaches 0 and 1. Congestion in stages 3 and 2 limits the traffic into stage 1 so the queue lengths cannot rise. Also note that the hotspot curves are much higher.

Stage 0: The hotspot queues are much longer than that for uniform traffic. For a high w , more packets arrive at stage 0 per cycle when p_0 approaches 0.5 from 0 or 1, because fewer packets are trapped in the queues in earlier stages due to saturation. Hence, queue lengths at stage 0 increase counter-intuitively when p_0 approaches 0.5. However, this local phenomenon is overshadowed by the larger drop in queue lengths at other stages.

3. *The effect of workload on queue lengths for $N = 16$ (Figure 3.7):* For uniform traffic, all graphs are similar to their counter-parts in Figure 3.5. For hotspot traffic, curves for stages 4 and 3 are similar to those for stages 3 and 2 in Figure 3.5. Queue lengths in stages 2 to 0 are much longer because when $N = 16$, more packets are sent to the hotspot for every w . The inverse-v shape of the curves for stages 0 and 1 when w is large is more apparent than that for $N = 8$.

Figure 3.5: Effect of workload (w) on queue lengths for $N = 8$.

Figure 3.6: Queue lengths at various stages (s) for $N = 8$.

Figure 3.7: Effect of workload (w) on queue lengths for $N = 16$.

4. *Queue lengths at various stages for $N = 16$ (Figure 3.8):* For uniform traffic, all graphs are similar to their counter-parts in Figure 3.6. For hotspot traffic, curves for stages 2 to 4 are also similar to their counter-parts in Figure 3.6. At stages 1 and 0, queues shorten when w is low and p_0 approaches 0.5 as expected. When w is high, deviation of p_0 from 0.5 causes saturation in earlier stages and reduces the packet arrival rate at stage 1, hence the inverse-v shaped curves.

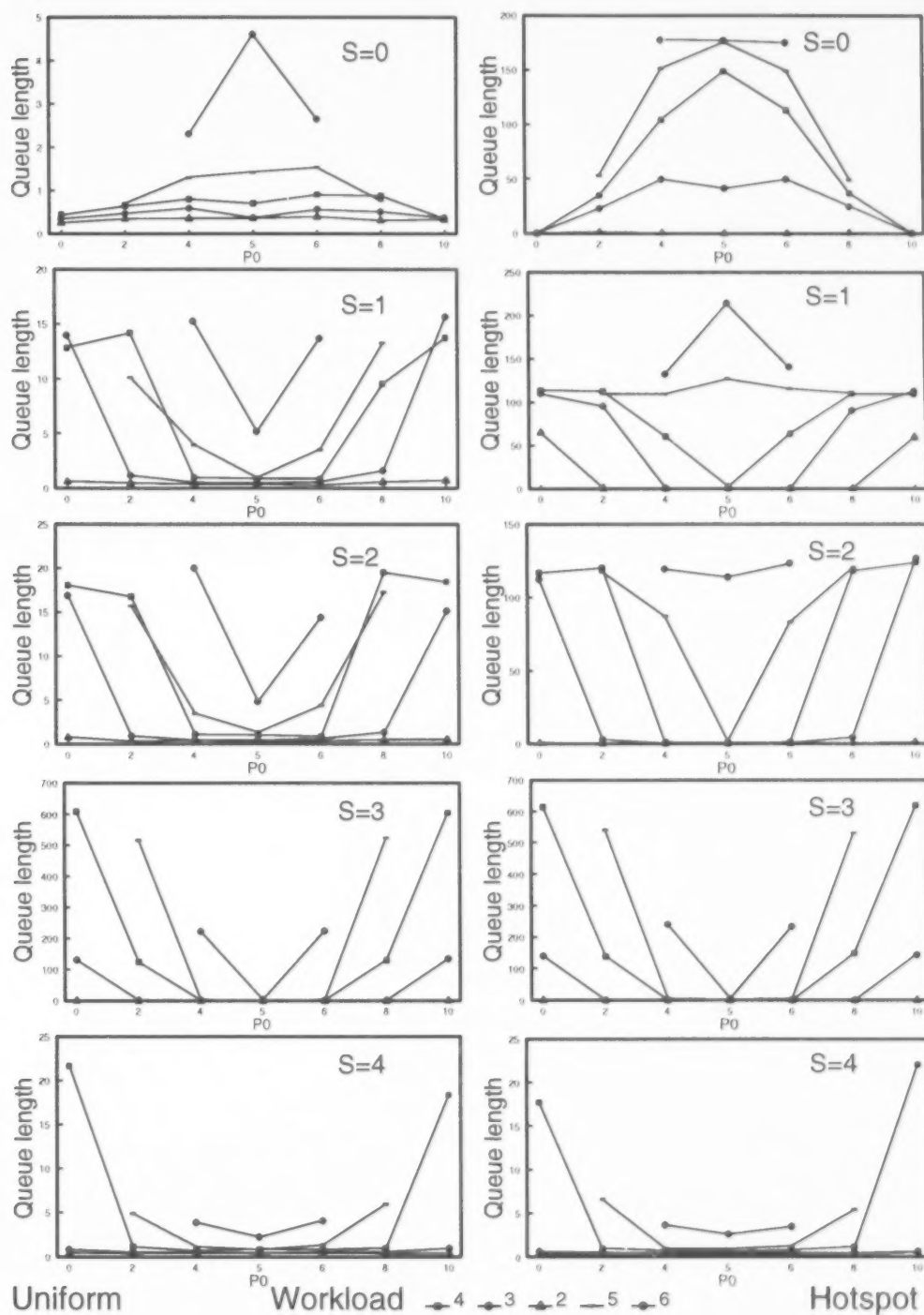
When the simulator was running in interactive mode, *tree saturation* (i.e., switches that are on the paths leading to the hotspot have noticeably longer queues than other switches) had been observed when hotspot traffic was combined with heavy workload. Figure 3.9 (brackets denote switches, and numbers in them are queue lengths) is a screen dump of a simulation run that has a hotspot at output 10. This phenomenon has been studied by other researchers [30, 37].

3.6 Concluding Remarks

We recursively defined a class of networks called k -GCNs and studied their symmetry properties. A path-selection problem on the 1-GCN was considered. We showed that using the 2 alternative paths with equal probabilities minimizes the average number of conflicts, and hence the average delay, for all traffic patterns if one global p_0 is used.

A simulation study verified our solution. Special features, such as tree saturation and the counter-intuitive increase in queue length at stage 0, were studied. Two possible extensions to the experiment are: 1. Try multiple hotspots, 2. Use distributions obtained from statistics of real applications.

We expect that for irregular networks, such as a 1-GCN with faulty switches, the path-selection problem would be very challenging. Some networks, say the ADM (§ 2.4.5), provide a different number of paths for different input/output pairs. It is anticipated that this approach (i.e., computing and then minimizing c) still works, although the analysis would be tedious.

Figure 3.8: Queue lengths at various stages (s) for $N = 16$.

```

===== ESC simulator v1.0 C1991 - by Alex C. Lam, UofA =====
  4      3      2      1      0
0- [ 0] 0 [ 10] 0 [ 0] 0 [ 1] 0 [ 3]
1- [ 5] 2 [ 1] 8 [ 1] 4 [ 1] 2 [ 4]
2- [ 1] 4 [ 5] 2 [ 0] 2 [ 2] 1 [ 1]
3- [ 6] 6 [ 3] 10 [ 0] 6 [ 6] 3 [ 1]
4- [ 1] 8 [ 0] 4 [ 4] 1 [ 2] 4 [ 1]
5- [ 0] 10 [ 1] 12 [ 1] 5 [ 3] 6 [ 1]
6- [ 4] 12 [ 3] 6 [ 6] 3 [ 1] 5 [ 2]
7- [ 4] 14 [ 4] 14 [ 2] 7 [ 0] 7 [ 3]
8- [ 3] 1 [ 28] 1 [ 35] 8 [ 315] 0 [ 2]
9- [ 1] 3 [ 13] 9 [ 135] 12 [ 312] 10 [ 1]
10- [ 3] 5 [ 3] 3 [ 85] 10 [ 375] 9 [ 577]
11- [ 4] 7 [ 3] 11 [ 61] 14 [ 336] 11 [ 559]
12- [ 7] 9 [ 0] 5 [ 43] 9 [ 0] 12 [ 5]
13- [ 8] 11 [ 2] 13 [ 88] 13 [ 5] 14 [ 5]
14- [ 1] 13 [ 4] 7 [ 91] 11 [ 2] 13 [ 1]
15- [ 3] 15 [ 3] 15 [ 36] 15 [ 1] 15 [ 2]

N: 16      S: 5
path-0 prob: 5
loading: 6
traffic: 2
stop at: 5000
hot spot: 10
hot prob: 18
time: 2009
queue length: 40.49
average delay: 205.89
pkt count: 19242
pkt gen.: 22494

```

```

time=>5000
[ESC]quit [f]ree run [a]lternate wise [u]lview switch [b]latch [t]imer [s]tatistics

```

Figure 3.9: Screen dump of simulation run that contains a hotspot at output 10.

Jean-Marie [58] adds extra stages to an indirect binary n -cube network to get multiple paths. However, extra stages and links must be added carefully so that the resulting network is roughly symmetrical with respect to the central stage. Queuing theory was applied to compute the optimal usage probability for each alternative path. The main conclusion is that for a k -extra-stage indirect binary n -cube network, assigning usage probability $\frac{1}{2^k}$ to each alternative path minimizes the overall average queue length in the network, regardless of the traffic pattern. Despite the similarity in the conclusions, noticeable differences distinguish our approach from theirs:

1. Although the GCN is equivalent to the indirect binary n -cube, no direct proof shows that after k stages are added to the indirect binary n -cube in the way described in [58], the resulting network is equivalent to a k -GCN. (Note that topological equivalence does not conserve on concatenation [124, Ch. 4])
2. The symmetry properties used in [58] to establish and solve the queuing equations are slightly different from ours. For instance, their topology must be symmetrical with respect to the central stage, while the k -GCN evidently lacks this symmetry. Moreover, our approach clearly exposes the role of each symmetry property, as well as its contribution to the final result. In [58], details

that offer insights into the network are hidden.

3. Our approach works even when the network under consideration does not offer some or all of the properties mentioned here or in [58]. We can always obtain an expression for c , although we may not be able to simplify it as we did before. Still, the equation $\frac{\partial c}{\partial p_{\alpha\beta k}} = 0$ can be solved numerically when an analytical solution is not easily obtainable.

We speculate that the average queue length in a k -GCN could be minimized by setting $p_{ij0} = \dots = p_{ij(2^k-1)} = \frac{1}{2^k}$, for all traffic patterns under some restrictions. A formal treatment of this generalization is a good intermediate term research goal. We also expect that Theorems 3.3 and 3.4 would hold for a broader class of MINs that possess certain symmetry properties. The settling of this issue seems to be an interesting long term goal.

Chapter 4

Permutation Routing on the ADM Network

This chapter presents a search-based, deterministic algorithm for routing permutations on the ADM network (§ 2.4.5) at compile time. If a permutation is realizable in 1 pass, the algorithm gives the required switch setting, otherwise, it is rejected. We formulate the routing problem in § 4.2 and then highlight some topological properties of the ADM. In § 4.3, the main idea of the algorithm is presented. We show how a permutation can be routed as a collection of 1-to-1 requests. In § 4.4, the performance of a sequential version of the algorithm is studied by simulation. Empirically, it takes almost linear time to process an arbitrary permutation. An upper bound derived in § 4.5 suggests that it takes at most $O(N \log N)$ time on average. No more than $O(N^3)$ time is needed in the worst case. Concluding remarks are given in § 4.6.¹

4.1 Introduction

This chapter presents a search-based, deterministic algorithm for routing permutations on the class of Augmented Data Manipulator (ADM) networks (§ 2.4.5). The

¹A preliminary version of this chapter was presented at the Sixth International Conference on Computing and Information in Peterborough, Canada [70].

algorithm is suitable for use by parallel compilers to generate communication efficient parallel code. If a permutation is *admissible* (realizable in 1 pass), the algorithm produces a list of *intermediate permutations* (defined in § 4.2), out of which the required switch setting can easily be computed. *Inadmissible* permutations are rejected. This algorithm is attractive because, empirically, the *average processing time* per permutation is almost linear in N (Figure 4.8).

We now give a brief overview of some existing results. *Banyan networks* (§ 2.4.3), such as the Ω network (§ 2.4.3) and the GCN (§ 2.4.3), provide a unique path for every input/output pair. One can route a permutation by trying to establish a path for every 1-to-1 request in the permutation [137]. This straight-forward approach takes $O(N \log N)$ time and space to route/reject a permutation. ($O(N)$ time for recognizing admissible permutations [52].) As pointed out in [45], the problem becomes difficult for multi-path networks, such as the ADM. Counting the number of distinct permutations admissible for multi-path networks is also non-trivial [45].

Adams and Siegel [2] give upper and lower bounds on the number of distinct permutations admissible by the ADM. These bounds diverge as N increases. Leland [77] presents recurrence equations for counting the number of distinct admissible permutations. Yet, no efficient characterization of the set of ADM-admissible permutations is known. Efficient permutation routing on the ADM remains an open problem. It is still unclear what complexity class the ADM permutation routing problem belongs to. By restricting to a particular control algorithm, Varma and Raghavendra [123] identify several commonly used permutation groups as admissible on the Gamma network (§ 2.4.5). But because the Gamma is strictly more powerful than the ADM, their findings are not directly applicable to the ADM. Three well-known schemes [112], namely the *Natural*, *Positive-Dominant*, and *Negative-Dominant* Routing Tags, can efficiently handle 1-to-1 requests on the IADM network (§ 2.4.5). If any of them is used for routing permutations, then the IADM accepts less permutations than the Ω network (details in [76]). The same conclusion holds for the ADM. If an efficient algorithm for routing all ADM-admissible permutations is unavailable, Ω -type net-

works will outperform the ADM in virtually all respects [76]. We attempt to address this issue.

If the connection pattern in each stage of links of the ADM is viewed as a bipartite graph, then a brute-force algorithm should include all possible maximum matchings for every stage in its search space. The resulting complexity would be exponential in N . (Theorem 4.8 states that stage 1 alone allows $Fib(N+1) + Fib(N-1) + 2$ maximum matchings [92], where $Fib(i)$ is the i th Fibonacci number: $Fib(1) = Fib(2) = 1$, $Fib(i+1) = Fib(i) + Fib(i-1)$.) Our first objective is to make the maximum number of matchings to be tried for any stage s dependent on s only, but not on N .

The next section gives more information on the ADM and introduces some notations. The routing problem is then formulated.

4.2 The Network, the Notation, and the Problem

A size N ADM consists of $n = \lg N$ stages of links, which interconnect $n + 1$ stages of switches (Figure 4.1). Stages of switches (links) are numbered from right to left starting from 0 (respectively, 1). Switches within a stage are numbered from top to bottom starting from 0. Denote switch j in stage i by $S[i, j]$. The ADM topology can be described using the following connections:

$$\text{At any stage } i, S[i, j] \text{ is connected to } \begin{cases} S[i-1, (j + 2^{i-1}) \bmod N] \\ S[i-1, j] \\ S[i-1, (j - 2^{i-1}) \bmod N] \end{cases} \quad \text{for } 0 < i \leq n$$

and $0 \leq j < N$. Modulus arithmetic is always used in dealing with switch labels so the "mod" will be omitted from now on.

We study the ADM topology from a new perspective. Let A_s refer to the interconnection pattern formed by the links in stage s . Then A_1 can be modeled by a bipartite graph on $2N$ vertices (Figure 4.3a), which has 1 connected component. The corresponding graph for A_2 has 2 connected components: one covering the odd vertices and the other covering the even vertices (Figure 4.3b). If the topology A_1 is called a *component* of type c_N (since it connects N input vertices to N output

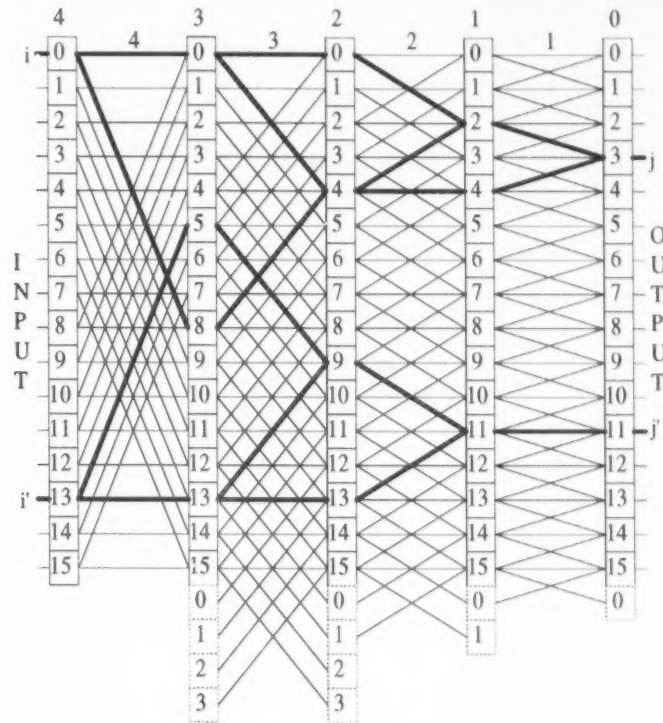


Figure 4.1: An ADM of size 16 (redundant links in stage 4 removed for simplicity).

vertices), then A_s has 2^{s-1} components of type $c_{N/2^{s-1}}$, where $1 \leq s \leq n$. Figure 4.3a illustrates the typical structure of any such component (switch numbers have been removed from the figure for simplicity). Nevertheless, one may verify that in any such component, the absolute difference between any 2 adjacent switch labels on one side of the component is constant, and is determined by the stage s in which the component resides.

As pointed out in [77], an ADM can be recursively divided into 3 parts: an odd sub-network, an even sub-network, and an output stage (Figure 4.2); where each sub-network is an ADM of size $\frac{N}{2}$. In the rest of this chapter, the ADM topology will be drawn the same way as before to avoid confusion. It should be noted that each of the 2 components in A_2 corresponds to the “stage 1” of the odd/even sub-network (Figure 4.2). Therefore, the “component” concept and the “odd/even sub-network” concept are convertible to each other.

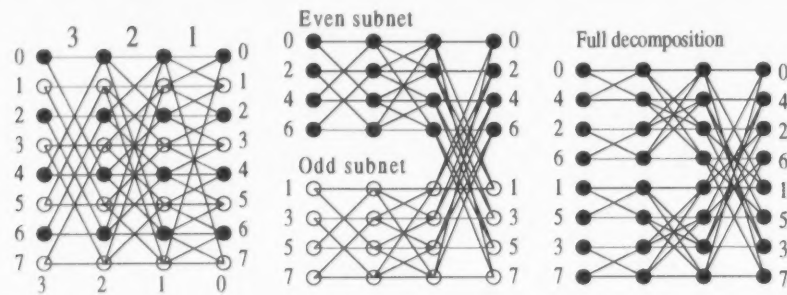
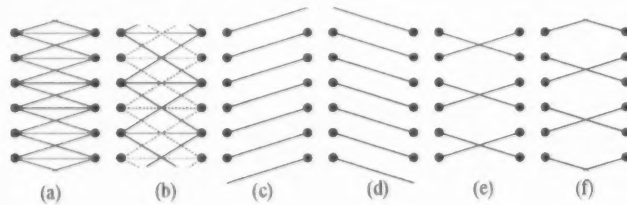


Figure 4.2: A size 8 ADM split into even and odd sub-networks.

Four basic permutations p_+ , p_- , p_r , and p_i (they originate from [2]; p_i is obtained by changing every upwards (downwards) connection in p_r to a downwards (upwards) connection) realizable by any component are graphically defined in Figures 4.3c-f. They are obviously subgraphs of c_N (Figure 4.3a). A connection request from input s to output d is abbreviated by (s, d) . We use ' $x \rightarrow y$ ' to mean 'switch x is routed to switch y .'

Figure 4.3: a) A component, b) 2 components, c) p_- , d) p_+ , e) p_r , f) p_i .

Assume the given permutation π is a vector of N elements $(v_0, v_1, \dots, v_{N-1})$, where $v_i = j$ means that input i of the network must be routed to output j . For $s = 1, 2, \dots, \lg N$, call p_s an *intermediate permutation* if it can be performed by the sub-network consisting of $A_s, \dots, A_1$ and all switches associated with them (i.e., the right-most s stages). In addition, define p_0 to be the identity permutation p_{id} . Now, permutation routing on the ADM can be formulated as follows: If a given permutation π is admissible, determine the sequence of intermediate permutations $p_1, \dots, p_n$ such that $p_n = \pi$. Figure 4.4 (in which switch labels have been removed for clarity) shows how an admissible permutation $\pi = p_4 = (2, 9, 12, 6, 11, 10, 1, 14, 5, 0, 4, 15, 13, 3, 7, 8)$ is routed on a size 16 ADM using the intermediate permutations $p_1, \dots, p_3$. Ulti-

mately, the algorithm must determine what partial permutation should be implemented by every component in every A_s in the ADM.

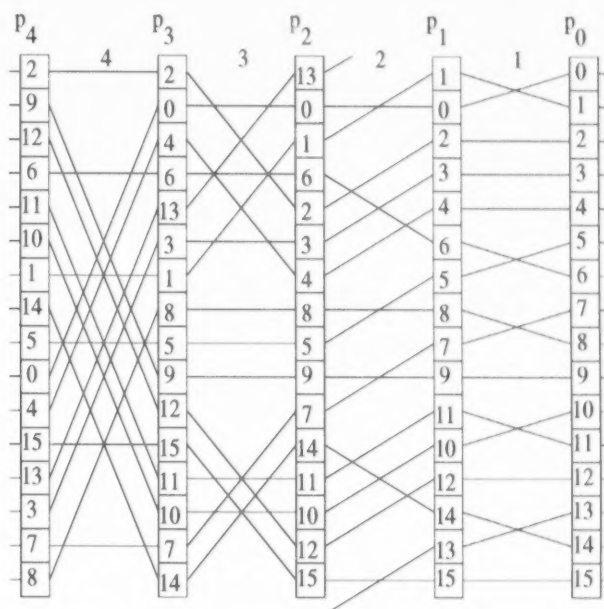


Figure 4.4: Intermediate permutations for a given admissible permutation.

4.3 Routing on the ADM

We start by presenting some properties of the ADM useful for routing a 1-to-1 request. Then, we show how a permutation can be routed as a collection of N such requests.

4.3.1 Routing One Connection Request

Finding all alternative paths for a given request (i, j) is a special case of a more general problem: Given any source switch in stage n and any target switch not in stage n , find all alternative paths between them, if any. Lemma 4.1 tackles this problem.

Lemma 4.1 *Given a source switch $S[n, i]$ in stage n and a target switch $S[s, j]$ in stage s , where $0 \leq s \leq n - 2$, exactly one of the following is true:*

1. all paths between the source and the target pass through the switch $S[s+1, j]$; or
2. no path passes through $S[s+1, j]$, but each of $S[s+1, j-2^s]$ and $S[s+1, j+2^s]$ is used by at least one path; or
3. there is no path between the source and the target.

Proof: It is not difficult to verify that the above 3 cases correspond to:

1. $|i - j| \bmod 2^{s+1} = 0$. (e.g., $S[4, 13] \rightarrow S[0, 11]$ in Figure 4.1)
2. $|i - j| \bmod 2^{s+1} \neq 0$ but $|i - j| \bmod 2^s = 0$. (e.g., $S[4, 0] \rightarrow S[1, 2]$ in Figure 4.1)
3. $|i - j| \bmod 2^{s+1} \neq 0$ and $|i - j| \bmod 2^s \neq 0$. (e.g., $S[4, 13] \rightarrow S[2, 8]$ in Figure 4.1)

We skip the details because it is a direct enumeration of the hardware. $\square$

When $s = n-1$, Lemma 4.1 still holds except that in case 2 the switches $S[s+1, j-2^s]$ and $S[s+1, j+2^s]$ are the same. All alternative paths for 2 sample connections (i, j) and (i', j') are given in Figure 4.1. Interested readers can see [90] for other recently found properties of the ADM. To abstract the routing requirement for each 1-to-1 request in a permutation, call the target switch $S[s, j]$ an *h-case switch* if case 1 of Lemma 4.1 holds for the request involving $S[s, j]$. Similarly, call $S[s, j]$ a *v-case switch* if case 2 holds.² Because every switch in stages $\lg N < s < 0$ can be in the alternative paths of several 1-to-1 requests in a given permutation, a switch can be a v-case for one request, but an h-case for another in the same permutation.

4.3.2 Permutation Routing

Given a permutation $\pi = p_n$ to be routed, we try to determine the sequence $p_1, p_2, \dots, p_{n-1}$ that can lead to the realization of π . We now provide details of a backtracking search process to solve the problem.

²The "h" in the term h-case switch stands for horizontal. The "v" in the term v-case switch refers to the geometrical layout of the paths.

The permutation implemented by any A_s is the result of merging all the partial permutations implemented by all the 2^{s-1} components in A_s . For $0 < s \leq n$, p_s is obtained by combining the permutation implemented by A_s with the intermediate permutation p_{s-1} realized by the sub-network formed by $A_{s-1}, \dots, A_1$ (and the associated switches). We now show that at most 4 candidate partial permutations (namely p_+ , p_- , p_r , and p_i as shown in Figure 4.3c-f) need to be tried for any component in any A_s , regardless of N and s .

Lemma 4.2 *Consider 2 adjacent switches in stage 1. If $S[1, i] \rightarrow S[0, i+1]$ and $S[1, i+1] \rightarrow S[0, i+2]$, then the only candidate for p_1 is p_+ . Similarly, if $S[1, i] \rightarrow S[0, i-1]$ and $S[1, i+1] \rightarrow S[0, i]$, then the only candidate for p_1 is p_- .*

Proof: Consider the p_+ case in Figure 4.5a. $S[0, i]$ is only accessible from $S[1, i-1]$ because $S[1, i] \rightarrow S[0, i+1]$ and $S[1, i+1] \rightarrow S[0, i+2]$. $S[1, i-1]$ must be routed to $S[0, i]$, otherwise, A_1 will not be implementing a permutation. Repeating this argument for $S[0, i-1], \dots, S[0, i+3]$ (wrap around if necessary) completes the proof. The p_- case is similar. $\square$

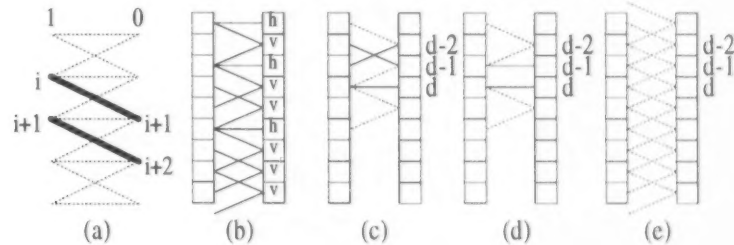


Figure 4.5: a) Case p_+ in A_1 . b-e) Various scenarios in A_1 .

The next lemma is a generalization of Lemma 4.2.

Lemma 4.3 *Consider 2 adjacent switches in some component $c_{N/2^{s-1}}$ in A_s , where $0 < s < n$. If $S[s, i] \rightarrow S[s-1, i+2^{s-1}]$ and $S[s, i+2^{s-1}] \rightarrow S[s-1, i+2^s]$, then the only candidate for this component is p_+ . Similarly, if $S[s, i] \rightarrow S[s-1, i-2^{s-1}]$ and $S[s, i+2^{s-1}] \rightarrow S[s-1, i]$, then the only candidate for this component is p_- .*

Proof: Similar to that of Lemma 4.2. Details omitted. $\square$

From § 4.3.1, every h-case (v-case) switch in stage 0 has 1 switch (2 switches) in stage 1 to choose from. Case 3 in Lemma 4.1 is ignored because all outputs are accessible from any input. A typical scenario at A_1 is shown in Figure 4.5b. The next 2 theorems are central to the proposed algorithm. We give a procedural proof for Theorem 4.4 to illustrate how the permutation routing algorithm will work.

Theorem 4.4 *Given a permutation π to be routed, the intermediate permutation p_1 has at most 4 candidates. In particular, exactly one of the following is true:*

1. p_1 has one unique candidate dictated by π ; or
2. p_1 may be one of p_+, p_-, p_r , and p_i , so further tests are necessary; or
3. π is an inadmissible permutation, i.e., p_1 has no candidate.

Proof: We show that the existence of even one h-case switch in stage 0 either eliminates all candidates for p_1 , or singles out 1 candidate. Suppose $S[0, d]$ is an h-case switch so it is required that $S[1, d] \rightarrow S[0, d]$. If $S[0, d-1]$ is also an h-case switch (Figure 4.5d), $S[1, d-1]$ must route to it. The next switch $S[0, d-2]$ can be considered similarly. This process repeats until a v-case switch is found or all switches in stage 0 have been considered.

Now assume $S[0, d]$ is an h-case switch and $S[0, d-1]$ is a v-case switch (Figure 4.5c). $S[1, d-2]$ must route to $S[0, d-1]$ since $S[1, d] \rightarrow S[0, d]$ due to the h-case. If:

1. $S[0, d-2]$ is an h-case switch (not in Figure 4.5c), then π is rejected (case 3 happens) because $S[1, d-2] \rightarrow S[0, d-1]$ so $S[0, d-2]$ cannot be reached. We call this situation a *conflict*.
2. $S[0, d-2]$ is a v-case switch (Figure 4.5c), then $S[1, d-1]$ must be routed to $S[0, d-2]$, because if $S[1, d-3] \rightarrow S[0, d-2]$, Lemma 4.2 requires $p_1 = p_+$.

We have considered $S[0, d-2]$; $S[0, d-3]$ can be treated in the same way as $S[0, d-1]$.

By repeating the above procedure, either exactly 1 candidate is identified, or π is rejected due to a conflict. Note that every switch in stage 0 actually has at most 1

switch in stage 1 to choose from. Now we have covered cases 1 and 3. Case 2 happens if there is no h-case switch in stage 0 (Figure 4.5e). We see 4 possibilities when 2 adjacent switches in stage 0 are considered:

1. $S[1, d+1] \rightarrow S[0, d]$ and $S[1, d] \rightarrow S[0, d-1]$: by Lemma 4.2, p_1 must be p_- .
2. $S[1, d+1] \rightarrow S[0, d]$ and $S[1, d-2] \rightarrow S[0, d-1]$: p_1 is p_r (p_i) if d is even (odd).
3. $S[1, d-1] \rightarrow S[0, d]$ and $S[1, d] \rightarrow S[0, d-1]$: p_1 is p_r (p_i) if d is odd (even).
4. $S[1, d-1] \rightarrow S[0, d]$ and $S[1, d-2] \rightarrow S[0, d-1]$: by Lemma 4.2, p_1 must be p_+ .

We have enumerated all cases. The proof is now complete. $\square$

Theorem 4.4 is generalized to give the next theorem.

Theorem 4.5 *Given a permutation π to be routed, every component in A_s , where $0 < s < n$, has at most 4 candidate partial permutations to be examined once p_{s-1} is known. In particular, in each component in A_s , exactly one of the following is true:*

1. *it must implement a unique partial permutation dictated by π and p_{s-1} ; or*
2. *it may implement one of p_+ , p_- , p_r , and p_i , so more tests are needed; or*
3. *a conflict occurs so no candidate can be selected for the component.*

Proof: Similar to that of Theorem 4.4. Details omitted. $\square$

Note that π can be rejected only if all candidates for p_{s-1} are rejected by case 3 for some s .

Corollary 4.1 *For any A_s , where $0 < s < n$, there are at most 2^{2^s} candidate permutations to be tried once p_{s-1} is known.*

Proof: By Theorem 4.5, each component has at most 4 candidate partial permutations. The 2^{s-1} components in A_s are all independent so their candidate partial permutations combine to give $4^{2^{s-1}} = 2^{2^s}$ candidate permutations. $\square$

Hence, an upper bound on the sequential time of the permutation routing algorithm is $O(N \prod_{s=0}^{n-1} 2^{2^s}) = O(N 2^{N-1})$ because it takes $O(N)$ time, as shown below, to check whether each candidate for p_{n-1} can lead to the realization of π .

```

/* cp is a candidate for  $p_{n-1}$ ,  $\pi$  is the given permutation */
proc IsRealizable( $cp, \pi$ : permutation;  $N$ : integer): boolean
  for  $i = 0$  to  $\frac{N}{2} - 1$  do /* modulo  $N$  arithmetics is used */
    if (not (( $i$ th element in  $cp = i$ th element of  $\pi$ ) and
      (( $i + \frac{N}{2}$ )th element in  $cp = (i + \frac{N}{2})$ th element of  $\pi$ )) or
      (( $i$ th element in  $cp = (i + \frac{N}{2})$ th element of  $\pi$ ) and
      (( $i + \frac{N}{2}$ )th element in  $cp = i$ th element of  $\pi$ ))) then
      return(false)
    endif
  endfor
  return(true)
endproc

```

4.3.3 The Routing Algorithm

Sequential tree-search now becomes a feasible solution to the routing problem because our findings limit the number of candidate partial permutations for any component to 4, independent of N and s . In particular, the root of the search tree is $p_0 = p_{id}$ while the leaves are candidates for p_{n-1} . Level s of the tree enumerates all candidate permutations for p_s , where $0 \leq s < n$. We try to determine p_1 , which (by Theorem 4.4) may be a unique permutation dictated by π , or one of p_+ , p_- , p_r , and p_i . Then we try to determine p_2 by first enumerating (by Theorem 4.5) the possible candidates for every component in A_2 based on the current p_1 . The partial permutations implemented by the components are combined with p_1 to give p_2 .

This process is applied to every A_s . Hence, the tree is traversed sequentially until a candidate for p_{n-1} that can lead to the realization of $p_n = \pi$ is found. Whenever a conflict occurs, it backtracks to explore the next available alternative at a higher level in the tree. π is rejected if no more candidates for p_{n-1} can be found. The correctness of this algorithm is evident. Although all main ideas of the algorithm have been presented, a pseudo code version is given below to enable easy implementation and validation.

```

proc main( $\pi$ : permutation;  $N$ : integer): boolean
  var plist: list-of-permutations
begin
  plist = findp(1,  $\log_2 N$ ,  $\pi$ ,  $p_{id}$ )

```

```

    if (plist = {}) then
        return(false) /* reject perm. */
    else
        print(plist) /* output intermediate perms. for admissible perm */
        return(true)
    endif
endproc

proc findp(i, input-stage: integer;  $p_{in}$ ,  $p_{right}$ : permutation): list-of-permutations
/* i is the current stage number */
    var  $p_{temp}$ : list-of-permutations; A: set;  $p_{try}$ ,  $p$ : permutation
begin
    if (i = input-stage) then /* terminates recursion */
        if realizable( $p_{right}$ ,  $p_{in}$ , N) then
            return({ $p_{in}$ })
        else
            return({})
        endif
    else /* recursive part */
        A = {} /* initialize working variable to empty set */
        for  $k = 0$  to  $2^{i-1} - 1$  do /* check all components in the current stage i */
            if  $\exists$  unique partial perm. for the  $k$ th component then /* Thm. 4.5 case 1 */
                put the unique partial perm. into  $p_{try}$ 
            else
                if a conflict is found then /* Thm. 4.5 case 3 */
                    return({}) /* backtrack */
                else /* Thm. 4.5 case 2 */
                    Add the  $k$ th component in stage i to A
                endif
            endif
        endfor
        if ( $|A| \neq 0$ ) then /*  $|A|$  is the cardinality of A */
            for each  $p$  in the  $4^{|A|}$  candidate partial perm. formed from A do
                Add  $p$  to  $p_{try}$  to form a complete perm.  $p_{try}$ 
                 $p_{temp} = \text{findp}(i + 1, \text{input-stage}, p_{in}, p_{try})$  /* search 1 level deeper */
                if ( $p_{temp} \neq \{\}$ ) then /* solution found, exit loop immediately */
                    return( $p_{temp} \cup \{p_{try}\}$ ) /* add candidate perm.  $p_{try}$  to return list */
                endif
            endfor
        else /*  $|A| = 0$ , only one candidate perm.  $p_{try}$  exists */
             $p_{temp} = \text{findp}(i + 1, \text{input-stage}, p_{in}, p_{try})$  /* search 1 level deeper */
            if ( $p_{temp} \neq \{\}$ ) then
                return( $p_{temp} \cup \{p_{try}\}$ ) /* add candidate perm.  $p_{try}$  to return list */
            endif
        endif
        return({}) /* backtrack */
    endif
endproc

```


endproc

If more than one switch setting can realize π , the first solution found is returned. The worst-case search tree (WCST) is illustrated in Figure 4.6. Note that nodes at the same level may have different branching factors. (For instance, not all nodes at level 1 have branching factor 4^2 because, say, if $p_1 = p_+$ gives only v-case switches in both components in A_2 , then each of $p_1 = p_i$ and p_r gives only v-case switches in one component and only h-case switches in the other component, while $p_1 = p_-$ gives entirely h-case switches in both components.) The existence of h-case switches will typically keep the search tree very small. The shape of the search tree is therefore highly data dependent. This algorithm can be slow on some “bad” permutations, but yet may perform well in practice.

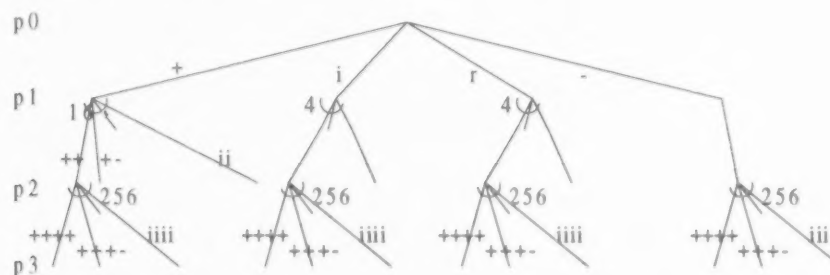


Figure 4.6: Worst case search tree.

This algorithm can easily be parallelized by searching the sub-trees simultaneously. Technical issues in optimizing the parallel version is out of our scope.

Note that all branchings in the search tree are due to the inability to decide which one of p_+ , p_- , p_r , and p_i should be used in components that contain v-case switches only. Leland [76, p. 70] indicates that no known method can make such a decision at stage s without looking ahead into the left side of stage s .³

³She also hints that using a backtracking algorithm to route permutations on the ADM would be “extremely inefficient” due to excessive backtracking [76, p. 80]. We found that it is not so bad.

4.4 Simulation Experiment

We implemented a sequential version of the above algorithm in C to study its performance. All simulation runs have been performed on DOS 486 machines. Such a single-user environment allows the elapsed real time to be accurately measured but speed and resources restrictions are tight. Following Leland's paper [77], we focused on networks of size $N = 2^n$, where $8 \leq N \leq 1024$. When $N = 8$, all the $8!$ permutations were tested. For every $N > 8$, we sampled at least 1 million distinct permutations. Each simulation run was performed 4 times. So, about 4 millions permutations were tested for each N because duplication (across runs) was negligible. The average number of admissible permutations over the 4 runs appears in Table 4.1.

Since the permutation generation method may bias the results, we briefly describe it here (as verified by test runs, the results are insensitive to the permutation generation method as long as it has enough randomness). The reader may also skip to § 4.4.1 now. The generation method is fast ($O(N)$ time, $N > 8$) and yet simple. Within each run, the generated permutations are random and distinct.

The i th element of a linear integer array $\mathcal{A}$ of size N is initialized to i , where $0 \leq i < N$. Then, random pairs of elements in $\mathcal{A}$ are swapped to give a random "seed" permutation. Find the largest integer k that satisfies $N(N-1)\cdots(N-k)\beta \geq 10^6$, where $1 \leq \beta < N-k$. Then set the i th element of a linear array MAP of size $N-k-2$ to i . The initialization is now done. The following recursive procedure is then invoked by the call $\text{gen}(N-1, k, \beta)$. A linear array B of size N will be filled in to give the desired random permutations. Note that MAP is slowly changing, which allows the randomness to be adjusted.

```

proc gen( $i, k, \beta$ )
  case  $i$ 
     $> N - k - 2$ : for  $j = 0$  to  $N - 1$ 
      if ( $\mathcal{A}[j] \geq 0$ ) call gen1( $i, j, k, \beta$ )
    endfor
   $= N - k - 2$ :  $l = 0$ 
    for  $j = 0$  to  $N - 1$ 
      if ( $\mathcal{A}[j] \geq 0$ ) and ( $l < \beta$ ) then
        call gen1( $i, j, k, \beta$ )

```

```

         $l = l + 1$ 
      endif
    endfor
    <  $N - k - 2$ : for  $j = 0$  to  $N - k - 3$ 
      Copy the  $j$ th non-negative element in  $\mathcal{A}$  to  $B[\text{MAP}[j]]$ 
    endfor
    Pass permutation  $B$  to the routing algorithm
    Randomly swap any 2 pairs of elements in MAP
  endcase
endproc

proc gen1( $i, j, k, \beta$ )
   $B[i] = \mathcal{A}[j]$ 
   $\mathcal{A}[j] = -1$ 
  gen( $i - 1, k, \beta$ )
   $\mathcal{A}[j] = B[i]$ 
endproc

```

4.4.1 Program Verification

The credibility of our sequential implementation is established in two ways:

1. For every permutation declared admissible by the program, an independent procedure checks whether the output list of intermediate permutations correctly implement the target permutation, based on the underlying hardware. Throughout the experiment, no admissible permutation failed the check. Note that it requires an exhaustive search, which is too expensive to incorporate into our experiment, to verify that a permutation is inadmissible.
2. The percentage of admissible permutations for each N is compared to existing analytical results. Leland's [77] recurrence equations count the number of distinct permutations admissible by the ADM. We fixed some apparent mistakes⁴ in [77] and used Maple V to generate the figures in column 3 (ADM_L) of Table 4.1. The number of permutations sampled in each run appears in column 5. Column 6 (ADM_E) gives the average number of admissible permutations reported by our algorithm. For $N = 8$, our result exactly matches Leland's. For

⁴We worked out a missing boundary condition and corrected several numerical errors. Caution: Leland's wrong numbers have been propagating in the literature (e.g., [120, p. 87 and p. 107]).

$N = 16$ and 32 , the measured percentages of admissible permutations (column 7) agree well with their theoretical counterparts (column 4). Percentage errors are $+3.85$ and $+2.68$, respectively. For every $N \geq 64$, the chance of hitting an admissible permutation is so small that all 4 million samples were rejected. The theoretical percentages of admissible permutations (column 4) support this argument.

N	$N!$	Leland's analysis		Our experiment		
		ADM_L	$\frac{ADM_L}{N!}$ (%)	Tested	ADM_E	$\frac{ADM_E}{Tested}$ (%)
8	40,320	26,496	65.714	40,320	26,496	65.714
16	2.092×10^{13}	1.549×10^{12}	7.404	1,048,320	80,609	7.689
32	2.631×10^{35}	1.169×10^{31}	4.443×10^{-3}	1,726,080	78.75	4.562×10^{-3}
64	1.268×10^{89}	3.243×10^{75}	2.558×10^{-12}	1,249,920	-	-
128	3.856×10^{215}	5.919×10^{177}	1.535×10^{-34}	1,007,872	-	-
256	8.578×10^{506}	1.110×10^{409}	1.294×10^{-94}	1,044,480	-	-
512	3.477×10^{1166}	1.237×10^{925}	3.558×10^{-238}	1,046,528	-	-
1024	5.419×10^{2639}	1.524×10^{2064}	2.846×10^{-572}	1,000,448	-	-

Table 4.1: Number of ADM-admissible permutations: Leland's versus our results.

Because every piece of code in the program have been exercised for $N = 16$ and 32 , we believe that its correctness extrapolates to all N .

4.4.2 Performance Study

The algorithm is evaluated from 2 perspectives. Since it is search-based, we first study the number of *nodes* searched (each node in the search tree corresponds to a candidate permutation attempted for an intermediate permutation p_s). Then, the practicality of the algorithm is measured by the *average processing time* (APT) per permutation.

Number of Nodes Searched

Average case: The average number of nodes visited before the algorithm rejects or admits a permutation (lower curve in Figure 4.7a) seems to approach 1.0 asymptotically. When N is large, most permutations sampled are inadmissible;

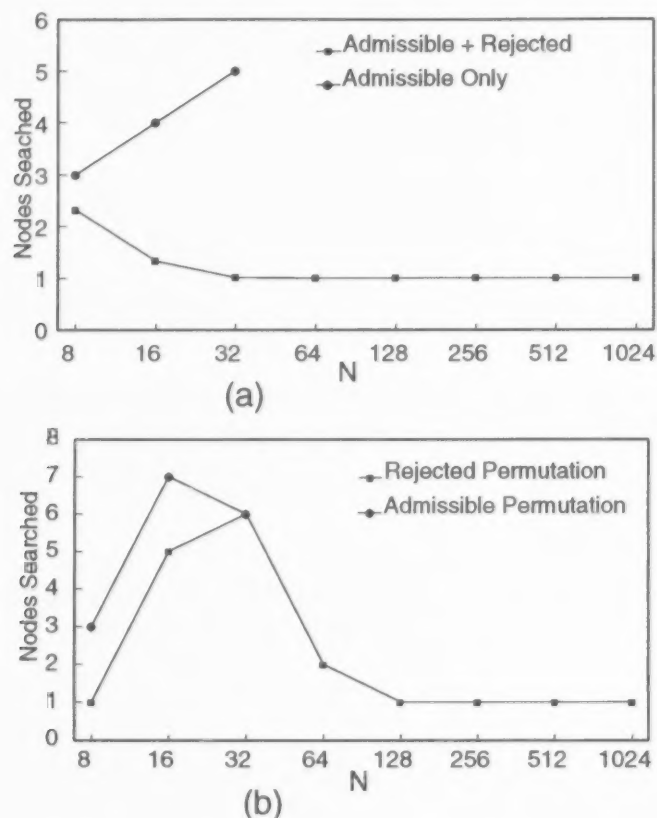


Figure 4.7: Average (a) and maximum (b) number of nodes searched per permutation.

apparently because no candidate can be found for p_1 . Although every admissible permutation needs at least n node-searches⁵ before it can be identified, the rapidly dropping percentage of admissible permutations has a dominating effect. Now we understand the lower curve. Despite the insufficient number of data points for admissible permutations (upper curve), we see that admitting a permutation rarely takes more than n node-searches.

Worst case: For each $N > 8$, the maximum number of nodes searched (Figure 4.7b) is taken over the corresponding 4 million samples. When N increases, the curves are expected to rise but the probability of hitting a “bad” case is so small that

⁵To keep the program simple and fast, we let $p_n (= \pi)$ be counted as an “intermediate permutation” also. In fact, only $n - 1$ nodes in the search tree are visited.

the curves apparently drop. Our sample size is still too small to allow definite conclusions to be drawn. For $N = 8$, remarkably, no permutation requires more than 1 node-search to reject or 3 node-searches to admit.

Real Elapsed Time: APT

The APT is computed by dividing the elapsed real time of each run by the number of permutations sampled. It includes overheads such as permutation generation and statistics collection. Because none of these tasks takes more than $O(N)$ time, the relation in Figure 4.8 is largely due to the routing algorithm. For small N 's, there is a counter-intuitive drop when N increases. To justify this, recall that an admissible permutation on average requires more node-searches to process than an inadmissible permutation (Figure 4.7a). The rapidly diminishing percentage of admissible permutations overshadows the increase in processing time due to the increase of N , and this accounts for the drop. But for each $N > 32$, all 4 million samples were rejected and each sample rarely needs more than 1 node-search (Figure 4.7b). Hence, the increase in processing time due to the doubling of N becomes a dominant factor so the curve rises. When the x-axis is drawn in linear scale, the algorithm seems to take $O(N)$ time to process a permutation when N is large.

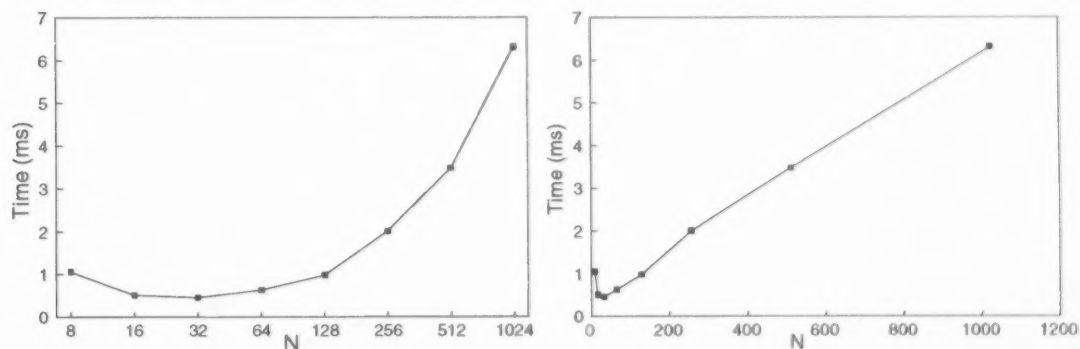


Figure 4.8: Average time to process each permutation; logarithmic and linear scale.

4.5 Approximate Time Complexity Analysis

To consolidate our confidence in the performance of the proposed algorithm, an upper bound on the *average number of nodes searched per permutation*, denoted $\bar{\mathcal{S}}_N$, is derived in this section. If U_N represents the set of all $N!$ possible permutations on $\{0, 1, \dots, N-1\}$, then by definition:

$$\bar{\mathcal{S}}_N = \frac{1}{N!} \sum_{p \in U_N} (\text{Number of node-searches required by } p). \quad (4.1)$$

Since every node-search takes $O(N)$ time, the average time complexity of the algorithm is $O(N\bar{\mathcal{S}}_N)$. Equation 4.1 is hard to evaluate because the exact relationship between a permutation and the number of node-searches it requires is not currently known. However, by classifying permutations into hierarchical sets and then computing the maximum number of node-searches required by any permutation in each set, an upper bound on $\bar{\mathcal{S}}_N$ can be obtained. Hence, the rest of this section mainly discusses the classification scheme, the size of each set, and the maximum number of node-searches for each set. At the end, we point out that if the search is done in a divide-and-conquer manner, the algorithm takes $O(N^3)$ time in the worst case.

4.5.1 Classification of Permutations

We will need the odd-even sub-network notion illustrated in Figure 4.2; but technical details, such as how the relabeling is done, do not concern us. We start our derivation by highlighting a fact that is evident from the discussion in § 4.3:

Observation 4.1 *Given a permutation π to be routed, the largest search tree arises only when all switches in stage 0 are v-case switches as seen by the algorithm.*

Due to the recursive nature of the ADM, a similar observation applies to any component in any stage. Hence, given a permutation π to be routed, the WCST arises only when all components in all stages just contain v-case switches.

The following definition plays a central role in our analysis. A *v-component* is one in which all switches in its right half are v-case switches. An *hv-component* is one in

which at least one h-case switch is present in its right half.

Definition 4.1 For any stage s , where $1 \leq s \leq n-1$, and integer t , where $0 \leq t \leq 2^{s-1}$, let W_t^s and R_s be sets of permutations defined by:

1. $W_t^s = \{w : w \text{ gives } t \text{ v-components and } 2^{s-1} - t \text{ conflict-free hv-components in } A_s; \text{ while each of } A_1, \dots, A_{s-1} \text{ contains conflict-free hv-components only}\}.$
2. $R_s = \{r : r \text{ gives only conflict-free hv-components in each of } A_1, \dots, A_{s-1}; \text{ but in } A_s \text{ a conflict occurs and causes } r \text{ to be rejected}\}.$

For $s = 1$, the set $\{A_1, \dots, A_{s-1}\}$ is empty.

For example, every permutation $w \in W_2^3$ is seen by the proposed algorithm to have unique candidates for p_1 and p_2 , but 4^2 candidates for p_3 . Note that each W_t^s contains both admissible and inadmissible permutations.

We are ready to present our classification scheme. By Theorem 4.4, U_N can be partitioned into 3 disjoint sets: W_0^1 , W_1^1 , and R_1 . They correspond to cases 1, 2, and 3 in Theorem 4.4, respectively. By applying Theorem 4.5 to A_2 , W_0^1 can be further partitioned into 4 disjoint sets described by:

W_0^2 : for every permutation $w \in W_0^2$, p_2 has a unique candidate because both components in A_2 are hv-components.

W_1^2 : for every permutation $w \in W_1^2$, p_2 has 4 candidates because one component in A_2 is a v-component while the other is an hv-component.

W_2^2 : for every permutation $w \in W_2^2$, p_2 has 4^2 candidates because both components in A_2 are v-components.

R_2 : for every permutation $r \in R_2$, no candidate for p_2 exists because at least one conflict (Theorem 4.5 case 3) occurs in any of the 2 components in A_2 .

By repeating a similar argument for all other stages in the network, a classification tree for all $N!$ permutations can be obtained. The case for $N = 16$ is illustrated in

Figure 4.9. We group the leaf nodes, which represent disjoint sets, into 2 sets:

$$R = W_0^{n-1} \cup \left(\bigcup_{i=1}^{n-1} R_i \right), \text{ and}$$

$$W_{\text{all}} = \bigcup_{i=1}^{n-1} \left(\bigcup_{j=1}^{2^{i-1}} W_j^i \right).$$

The motivation for this grouping is: every permutation $r \in R$ needs at most $n - 1$

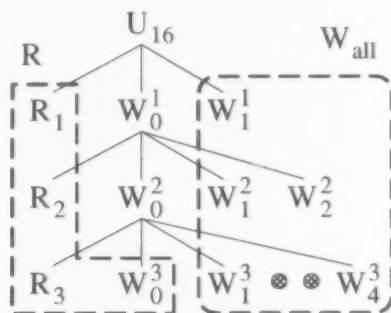


Figure 4.9: A classification of permutations for $N = 16$.

node-searches to admit/reject; and this allows us to focus on the W_j^i 's in W_{all} . If $\mathcal{N}(W_j^i)$ denotes the maximum number of node-searches required by any permutation $w \in W_j^i$, then an upper bound on $\overline{\mathcal{S}}_N$ (Equation 4.1) can be written as follows:

$$\bar{\mathcal{S}}_N \leq \frac{1}{N!} \left\{ (n-1) \cdot |R| + \sum_{i=1}^{n-1} \left[\sum_{j=1}^{2^{i-1}} \mathcal{N}(W_j^i) \cdot |W_j^i| \right] \right\}. \quad (4.2)$$

4.5.2 Cardinality of W_t^s

We first characterize W_1^1 and compute its cardinality, then the result is generalized to other possible sets W_ℓ^s . If $w(i)$ denotes the image of i under the permutation w , then W_1^1 can be characterized by:

Lemma 4.6 $W_1^1 = \{w : |i - w(i)| \bmod 2 = 1, \forall i \in \{0, 1, \dots, N-1\}\}.$

Proof: Without loss of generality, let i and $w(i)$ be even and odd numbers, respectively. No path from input i to output $w(i)$ can use any odd switch except at stage 0 because $S[n, i]$ is in the even sub-network. Only 2 even switches in stage 1 have access

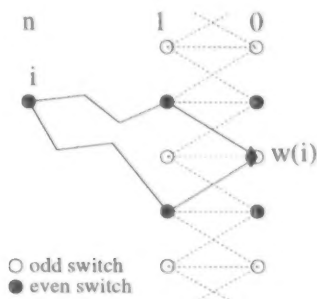


Figure 4.10: A v-case switch in stage 0.

to the odd switch $S[0, w(i)]$. So $S[0, w(i)]$ must be a v-case switch (Figure 4.10). Since i is arbitrary, this lemma is true for all $i \in \{0, 1, \dots, N-1\}$. $\square$

We are now ready to compute the cardinality of W_1^1 .

Lemma 4.7 $|W_1^1| = [(\frac{N}{2})!]^2$.

Proof: From Lemma 4.6, any permutation w that only maps even inputs to odd outputs, and only odd inputs to even outputs, would be in W_1^1 . Figure 4.11a, which shows the first and last stages of an ADM in the form of a bipartite graph, depicts all the possible connections. It has 2 connected components as shown in Figure 4.11b and c. Each of them has $\frac{N}{2}$ pairs of nodes and allows all $(\frac{N}{2})!$ maximum matchings. Hence, the total number of permutations allowed by Figure 4.11a is $[(\frac{N}{2})!]^2$. $\square$

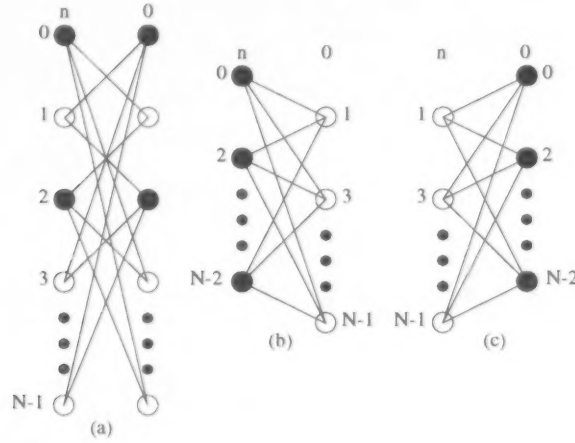
Existing results from 2 papers are needed to generalize Lemma 4.7 to any possible W_t^s . If Θ_N denotes the set of permutations realizable by A_1 , then O'Donnell and Smith have shown:

Theorem 4.8 [92]: $|\Theta_N| = \text{Fib}(N+1) + \text{Fib}(N-1) + 2$.

Let $\mathcal{A}^N/j$ denote the set of permutations realizable by a size N ADM whose A_1 always implements permutation j . The next theorem is due to Adams and Siegel:

Theorem 4.9 [2, Lem. 7]: For any $j, h \in \Theta_N$, if $j \neq h$ and $j \notin \{p_+, p_-, p_r, p_i\}$, then $\mathcal{A}^N/j$ and $\mathcal{A}^N/h$ are disjoint.

The proof in [2] for Theorem 4.9 allows us to make the following stronger statement:

Figure 4.11: Bipartite graphs showing stage 0 and n .

Corollary 4.2 Let σ_1 and σ_2 be the unique candidates (i.e., $\sigma_1 \notin \{p_+, p_-, p_r, p_i\}$ and $\sigma_2 \notin \{p_+, p_-, p_r, p_i\}$) for p_1 when the given permutations w_1 and w_2 are to be routed, respectively. If $\sigma_1 \neq \sigma_2$, then w_1 and w_2 are distinct.

We start by defining a set of permutations $\Phi_N = \{\theta : \theta \in \Theta_N \text{ and } \theta \text{ gives an hv-component in } A_1\}$. Since only 4 permutations in Θ_N , namely p_+ , p_- , p_i , and p_r , do not give any h-case switch in stage 0, Theorem 4.8 implies that $|\Phi_N| = |\Theta_N| - 4 = \text{Fib}(N+1) + \text{Fib}(N-1) - 2$. Let $\phi_N = |\Phi_N|$ for short. Lemma 4.7 is generalized to:

Theorem 4.10 For any stage $1 \leq s \leq n-1$ and integer $1 \leq t \leq 2^{s-1}$,

$$|W_t^s| = C_t^{2^{s-1}} \cdot \alpha \cdot \nu^t \cdot \mu^{2^{s-1}-t},$$

where C_b^a denotes the binomial coefficient $\frac{a!}{b!(a-b)!}$,

$$\alpha = \alpha(N, s) = \prod_{i=1}^{s-1} \left(\phi_{\frac{N}{2^{i-1}}} \right)^{2^{i-1}},$$

$$\nu = \nu\left(\frac{N}{2^{s-1}}\right) = \left[\left(\frac{N}{2^s}\right)!\right]^2, \text{ and}$$

$$\mu = \mu\left(\frac{N}{2^{s-1}}\right) = \phi_{\frac{N}{2^{s-1}}} \left[\left(\frac{N}{2^{s+1}}\right)!\right]^2.$$

Proof: When $s = t = 1$, the above equation reduces to Lemma 4.7 if $\prod_{i=1}^0 (\dots) = 1$. In general, $\alpha(N, s)$, which denotes the total number of possible settings for the sub-network that includes $A_1, \dots, A_{s-1}$, can be computed using Theorem 4.8, Corollary

4.2, and the recursive nature of the network. The result is $\prod_{i=1}^{s-1} (\phi_{\frac{N}{2^{i-1}}})^{2^{i-1}}$, where $(\phi_{\frac{N}{2^{i-1}}})^{2^{i-1}}$ is the number of possible settings for A_i . Since exactly t out of the 2^{s-1} components in A_s are v-components, there are $C_t^{2^{s-1}}$ possible selections of these t v-components. The corresponding sub-network of each of these t v-components allows $\nu(\frac{N}{2^{s-1}}) = \{[\frac{(\frac{N}{2^{s-1}})}{2}]\}^2$ possible permutations on $\{0, 1, \dots, \frac{N}{2^{s-1}} - 1\}$ due to Lemma 4.7. So these t sub-networks combine to give $[(\frac{N}{2^s})!]^{2t}$ possibilities. Each of the remaining $2^{s-1} - t$ hv-components in A_s has a unique candidate, so the corresponding sub-network of each component allows $\mu(\frac{N}{2^{s-1}}) = \phi_{\frac{N}{2^{s-1}}} [(\frac{N}{2^{s+1}})!]^2$ possible permutations on $\{0, 1, \dots, \frac{N}{2^{s-1}} - 1\}$. These $2^{s-1} - t$ sub-networks are independent so $(\phi_{\frac{N}{2^{s-1}}})^{2^{s-1}-t} [(\frac{N}{2^{s+1}})!]^{2^{s-2t}}$ possibilities are generated. $\square$

4.5.3 Number of Nodes Searched

This sub-section first gives the size of the WCST for an ADM with N inputs, then $\mathcal{N}(W_t^s)$, that is the maximum number of node-searches required by any permutation $w \in W_t^s$, is computed. Let T_s denote the number of leaf nodes in a s -level WCST:

Theorem 4.11 $T_0 = 1$. $T_1 = 4$. For $l \geq 2$, $T_l = [T_{l-1} + (T_{l-2})^2]^2$.

Proof: Since $p_0 = p_{id}$ (the identity permutation), $T_0 = 1$ follows. Theorem 4.4 implies that $T_1 = 4$. By manually generating the WCST (see for example Figure 4.6), we found that $T_2 = 4^2 + 2 \cdot 4 + 1 = (4 + 1)^2$. Similarly, T_3 was found to be $((4 + 1)^2 + 4^2)^2$. Now we see that $T_l = (T_{l-1} + (T_{l-2})^2)^2$ is a candidate for the general equation. Again, by manually generating the WCST, both $T_4 = (((4 + 1)^2 + 4^2)^2 + (4 + 1)^4)^2$ and $T_5 = [(((4 + 1)^2 + 4^2)^2 + (4 + 1)^4)^2 + ((4 + 1)^2 + 4^2)^4]^2$ were found to agree with the candidate equation.

To verify that the candidate solution precisely matches the recursive model of the ADM for general l , expand T_l to $T_l = T_{l-1}^2 + T_{l-2}^2 T_{l-1} + T_{l-1} T_{l-2}^2 + T_{l-2}^4$. A property of the WCST mentioned in brackets on page 66 will be used. Recall that the p_1 of an ADM sub-network that gives a WCST, which has T_l leaf nodes, has 4 candidate permutations. Suppose that when $p_1 = p_+$, the " A_1 " of the two sub-networks are

v-components so they each has 4 candidate permutations (Figure 4.12a). This case gives the T_{l-1}^2 term in T_l . When $p_1 = p_i$, the “ A_1 ” of the even sub-network still has 4 candidate permutations as before but the “ A_1 ” of the odd-network contains h-case switches only so it has 1 unique candidate permutation (Figure 4.12b). This case gives the $T_{l-2}^2 T_{l-1}$ term in T_l . Symmetrically, Figure 4.12c corresponds to the term $T_{l-1} T_{l-2}^2$. When $p_1 = p_-$, the “ A_1 ” of both sub-networks contain h-case switches only (Figure 4.12d). This case gives the T_{l-2}^4 term in T_l . $\square$

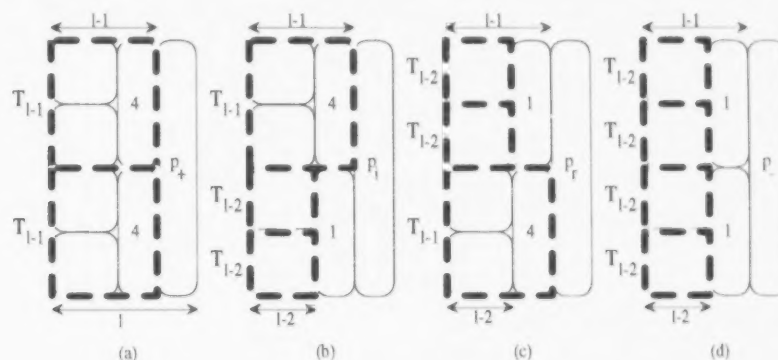


Figure 4.12: Explanation for the terms in T_l .

Let Γ_s denote the number of nodes, except the root, in a s -level WCST. For $s > 0$,

$$\Gamma_s = \sum_{i=1}^s T_i. \quad (4.3)$$

For convenience, define $\Gamma_0 = 1$. Now, $\mathcal{N}(W_t^s)$ in Equation 4.2 can be computed by:

Theorem 4.12 For $1 \leq s \leq n-1$ and $1 \leq t \leq 2^{s-1}$,

$$\mathcal{N}(W_t^s) = (s-1) + (\Gamma_{n-s})^t [(\Gamma_{n-s-1})^2]^{2^{s-1}-t}.$$

Proof: As each of $p_1, \dots, p_{s-1}$ has a unique candidate for any $w \in W_t^s$, no backtracking involves these stages so $s-1$ node-searches are needed. In A_s , the corresponding sub-network of each of the t v-components requires at most a WCST of size Γ_{n-s} . These t sub-networks give $(\Gamma_{n-s})^t$ nodes to be examined in total (Corollary 4.2). In the corresponding sub-network of each of the remaining $2^{s-1} - t$ hv-components in A_s , the maximum number of possibilities is $(\Gamma_{n-s-1})^2$, because we assume each of

the 2 sub-networks in each of the $2^{s-1} - t$ sub-networks in A_s gives a WCST. These $2^{s-1} - t$ sub-networks in A_s combine to generate $[(\Gamma_{n-s-1})^2]^{2^{s-1}-t}$ nodes. $\square$

4.5.4 An Upper Bound

Equation 4.2 now becomes:

$$\begin{aligned}\bar{\mathcal{S}}_N \leq \widetilde{\mathcal{S}}_N &= (n-1)\left(1 - \frac{|W_{\text{all}}|}{N!}\right) + \frac{1}{N!} \sum_{i=1}^{n-1} \left[\sum_{j=1}^{2^{i-1}} \mathcal{N}(W_j^i) \cdot |W_j^i| \right] \\ &= (n-1)\left(1 - \frac{1}{N!} \sum_{i=1}^{n-1} \sum_{j=1}^{2^{i-1}} |W_j^i| \right) + \frac{1}{N!} \sum_{i=1}^{n-1} \left[\sum_{j=1}^{2^{i-1}} \mathcal{N}(W_j^i) \cdot |W_j^i| \right] \quad (4.4)\end{aligned}$$

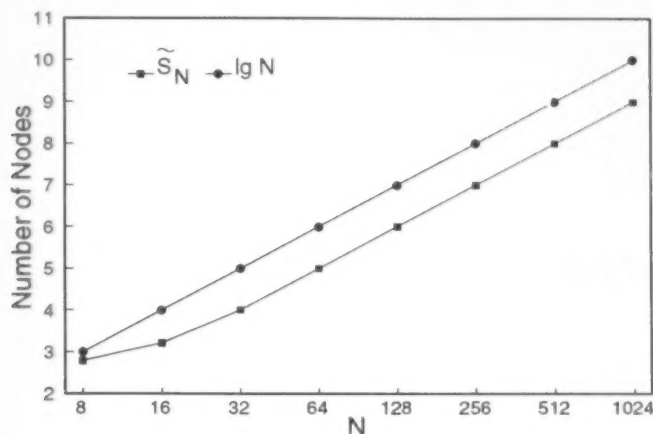
Again, we used Maple V (source code in Appendix C) to evaluate Equation 4.4. By varying N from 8 to 1024, we obtained the results shown in columns 1 and 2 of Table 4.2 (other columns will be discussed shortly). When plotted in logarithmic

N	$\widetilde{\mathcal{S}}_N$	$Z = \frac{\Gamma_{n-1} \cdot W_1^1 }{N!}$	T_{n-1}	$ W_1^1 $
8	2.787500000	.4142857143	25	576
16	3.212169790	.1328671329	1681	1625702400
32	4.009019977	.8849641560x10 ⁻²	5317636	.4377631367x10 ²⁷
64	5.000036187	.3618577238x10 ⁻⁴	.6631491470x10 ¹⁴	.6923783735x10 ⁷¹
128	6.000000000	.3735803750x10 ⁻⁹	.8947678120x10 ²⁸	.1610029356x10 ¹⁷⁹
256	7.000000000	.3087342589x10 ⁻¹⁹	.1780982607x10 ⁵⁷	.1487031564x10 ⁴³²
512	8.000000000	.1410341953x10 ⁻³⁹	.6664617483x10 ¹¹³	.7358513357x10 ¹⁰¹⁴
1024	9.000000000	.2159151111x10 ⁻⁸⁰	.9675705744x10 ²²⁶	.1209154431x10 ²³³⁴

Table 4.2: Numerical results produced by Maple V.

scale (Figure 4.13), $\widetilde{\mathcal{S}}_N$ seems to be consistently lower than $\lg N$ (algebraically proving that $\widetilde{\mathcal{S}}_N = O(\log N)$ appears to require more participation from the mathematics community). Therefore, we expect that the actual average time complexity of the ADM permutation routing problem is likely to lie between $O(N)$ and $O(N \log N)$; probably closer to $O(N)$ as suggested by our simulation.

Now we provide an intuitive account of the results. It is evident from Figure 4.13 that the first term in Equation 4.4 dominates (this agrees with simulation results in Figures 4.7 and 4.8). To reason why the second term diminishes with N , we monitored the quantity $Z = \frac{\Gamma_{n-1} \cdot |W_1^1|}{N!}$ in Table 4.2, which contributes most to the

Figure 4.13: $\widetilde{S}_N$ and $\lg N$ versus N .

summation because W_1^1 is the largest W_t^s and the corresponding Γ_{n-1} is also the largest. Note that Z decreases quickly when N increases, especially for large N 's. Also, T_{n-1} roughly squares whenever N doubles (note: one can prove by induction that $\Gamma_{s-1} \leq 2T_{s-1}$, $0 \leq s < n$). $|W_1^1|$ grows faster but less than cubically.

The bound can be tightened in the future by removing 3 pessimistic assumptions.

1. Most r 's in R take 1 instead of $n - 1$ node-searches to reject (Figure 4.7).
2. Most w 's in W_1^1 do not need Γ_{n-1} node-searches (many inadmissible permutations can be quickly rejected). This argument applies to other W_t^s , $t \geq 1$.
3. The actual search typically does not visit all nodes in the WCST, especially when there are multiple solutions.

This analysis has also opened the door to improving the performance of the algorithm. Lemma 4.6 implies an $O(N)$ time procedure for identifying the "worst" permutations. For instance, the "shift-down-by-1" permutation $w(i) = (i + 1) \bmod N$ gives the full WCST but there is an obvious way to route w : use downwards links in A_1 and horizontal links in all other A_s . The "shift-up-by-1" case is symmetrical.

4.5.5 Worst-Case Performance

We now show that the search process can be restructured so that the resulting algorithm takes $O(N^3)$ time in the worst case. Recall that the above algorithm (procedure `findp()`, to be exact) invokes itself recursively to examine A_{s+1} only if all components in A_s are conflict-free (i.e., at least 1 candidate for p_s exists). This policy takes advantage of the observation that most permutations are inadmissible and they are typically rejected due to conflicts at right-most stages of the network.

In the restructured algorithm, if the A_1 of an ADM is conflict-free (i.e., it has 1 or 4 candidates), then partial permutations of length $\frac{N}{2}$ need to be routed by the even and odd sub-networks. This routing can be done by 2 recursive calls to the algorithm itself. In so doing, backtracking in one sub-network no longer affects the other sub-network. Let T_N denote the time required by the new algorithm to process an arbitrary permutation for a size N ADM. Then, we have the following recurrence:

$$T_N \leq \alpha N + 4(2T_{\frac{N}{2}}),$$

where α is a constant, αN represents all $o(N)$ computations in the body of the algorithm, and $4(2T_{\frac{N}{2}})$ is the maximum time spent on recursive calls. The boundary condition $T_1 = 1$ gives the closed form solution: $T_N \leq \alpha N^{\frac{4^n-1}{3}} + 8^n$, where $n = \log_2 N$. It simplifies to $T_N \leq \alpha N^{\frac{N^2-1}{3}} + N^3$, which means $T_N = O(N^3)$.

In comparing the two algorithms, note that a conflict in A_1 of the odd sub-network may nullify the work done in the even sub-network because the advantage mentioned at the beginning of this sub-section is lost. Hence, the divide-and-conquer approach does not necessary perform better on the average.

4.6 Conclusion

We presented a deterministic, backtracking algorithm for routing permutations on the ADM at compile time. It is much faster than brute-force search because the maximum number of permutations to be examined for any A_s is independent of N .

The algorithm is expected to be easy to parallelize. A sequential version performed well in our experiment because “bad” cases are rare, especially for large N 's. Although there may still be room for further speedup, large improvement is impossible because the current version apparently takes linear time on average. Analysis suggests that the algorithm takes at most $O(N \log N)$ time on average. The worst-case analysis shows that the ADM permutation problem can be solved in polynomial time.

It is instructive to compare the ADM with the Ω network in Table 4.3. Although the ADM is slightly more expensive, both MINs have $O(N \log N)$ hardware cost. Leland [77] shows that the number of admissible permutations are roughly $(N!)^{0.694}$ and $(N!)^{0.5}$, respectively (Table 4.4). The superior combinatorial power of the ADM had not been fully exploited because no polynomial routing algorithm was known. In contrast, the Ω just requires $O(N)$ sequential time [52] to recognize an admissible permutation, and $O(N \log N)$ time [137] (both average and worst case) to route an admissible permutation. Our algorithm, which is likely to take $O(N \log N)$ average time to process an arbitrary permutation and never needs more than $O(N^3)$ time, has revived the ADM as an attractive alternative to the Ω .

	Hardware cost	Admissible permutations	Routing complexity
Ω	$O(N \log N)$	$\approx (N!)^{0.5}$	$O(N \log N)$
ADM	$O(N \log N)$	$\approx (N!)^{0.694}$	$O(N \log N)$ (conjecture)

Table 4.3: Comparison of the ADM and Ω networks.

N	$N!$	$(N!)^{0.694}$	$(N!)^{0.5}$
8	40320	1571.2	200.8
16	$.2092 \times 10^{14}$	$.1755 \times 10^{10}$	$.4574 \times 10^7$
32	$.2631 \times 10^{36}$	$.3815 \times 10^{25}$	$.5129 \times 10^{18}$
64	$.1268 \times 10^{90}$	$.6882 \times 10^{62}$	$.3562 \times 10^{45}$
128	$.3856 \times 10^{216}$	$.4138 \times 10^{150}$	$.6209 \times 10^{108}$
256	$.8578 \times 10^{507}$	$.6483 \times 10^{352}$	$.2928 \times 10^{254}$
512	$.3477 \times 10^{1167}$	$.3798 \times 10^{810}$	$.1864 \times 10^{584}$
1024	$.5418 \times 10^{2640}$	$.9447 \times 10^{1832}$	$.7361 \times 10^{1320}$

Table 4.4: $N!$, $(N!)^{0.694}$, and $(N!)^{0.5}$ versus N .

The algorithm can generate statistical information (e.g., Figure 4.7) which may help us study ADM-admissible permutations. The algorithm can verify/calibrate new

heuristics/algorithms. It also provides a basis out of which efficient heuristics can be evolved. A detailed analysis of its exact average time complexity appears to be a challenging research topic. By carefully combining the divide-and-conquer approach (§ 4.5.5) with the current approach, it is hoped that the average time complexity of the algorithm can be proven to be between $O(N \log N)$ and $O(N)$.

Chapter 5

On a Class of k -Nonblocking Networks

In § 5.1, we point out that although high performance multiprocessor systems demand nonblocking networks, cost considerations typically force the architect to adhere to rearrangeable or even blocking networks. To enable a better balance between cost and performance, we introduce the general notion of k -nonblocking networks in § 5.2, where k is the number of simultaneous connections that will not be blocked. The k -nonblockingness of some existing networks are studied. A systematic scheme for constructing strict-sense k -nonblocking networks, based on Beneš' rearrangeable network, is given in § 5.3. The constructed network (with $k = N$) is cheaper than an existing strictly nonblocking network. Efficient sequential and parallel routing algorithms for the constructed network are presented in § 5.4. The hardware cost of the network is computed in § 5.5. § 5.6 contains some concluding remarks.¹

¹A preliminary version of this chapter was presented at the Eighth Annual Canadian High Performance Computing Symposium in Toronto, Canada [72].

5.1 Introduction

The traditional approach to minimizing the cost of the INs in SIMD machines is to use low cost blocking networks that support the traffic patterns generated by typical applications (e.g., using the Ω network for matrix manipulation [73]). When a new application is ported to the machine, the programmer is required to analyze and re-schedule the communication patterns inherent in the application. This difficult task restricts the usefulness of the machine. On the other hand, multi-user MIMD machines (especially real-time systems) require nonblocking networks because the communication pattern is not known in advance.

Although high performance systems demand nonblocking networks, cost considerations typically force the architect to adhere to rearrangeable or even blocking networks. For instance, the nonblocking network proposed by Lin and Pippenger in [79], which is derived from Beneš' rearrangeable network (§ 2.4.4), is over 100 times more expensive than the latter when the size N is only 128. This cost gap widens with increasing N ([99] has a theoretical study). However, rearrangeable and blocking networks are too restricted in power.

Since the communication demand on a typical multiprocessor system is low most of the time (i.e., most of the parallel tasks are running independently), a full fledged nonblocking network is unnecessary. We introduce the general notion of k -nonblockingness to allow the high cost of a nonblocking network and the long delay of a rearrangeable/-blocking network to be balanced by a good choice of k . By giving a systematic scheme for constructing k -nonblocking MINs in the strict sense, and providing efficient routing algorithms, we show that the nonblocking operation can be achieved at a lower cost. The cost gap between blocking and nonblocking networks is now interpolated by a spectrum of new multi-path MINs. We start by defining k -nonblocking networks.

5.2 k -Nonblockingness

This chapter focuses on MINs with an equal number of *input* and *output* terminals. A MIN with N inputs has *size* N . We assume circuit-switching operation with 1-to-1 requests only. MINs are made of 2x2 switches unless stated otherwise. Incoming requests are handled one at a time.

5.2.1 Definition

We first define k -nonblocking MINs, and then give examples.

Definition 5.1 *A MIN η of size N is k -nonblocking, where $1 \leq k \leq N$, if η never blocks an incoming request whenever there are less than k established connections.*

Evidently, k -nonblockingness is *workload*-dependent (workload is defined to be the number of established connections). When the workload exceeds k , the MIN may still be nonblocking if the traffic pattern is “good,” otherwise, it is rearrangeable (e.g., the proposed construction scheme in § 5.3) or blocking (e.g., the Clos network in Theorem 5.3 with $m < n$). Even when the traffic pattern is “bad,” rearrangement/blocking can be eliminated by dividing each communication step involving more than k connections into *phases*, each involving k or less requests. Clearly, the connection power of a k -nonblocking MIN grows monotonically with k . Combining with Beneš’ classification (§ 2.3.1), we can have wide- and strict-sense k -nonblocking networks.

Although $k = 0$ is excluded from Definition 5.1, we can define those networks lacking the *full access* (§ 2.4.1) capability to be 0-nonblocking, that is, a request can be blocked even when the network is idle. A single stage shuffle-exchange (SE) network (§ 2.4.2) of size $N > 2$ is an example. In general, any size N MIN with less than $\lg N$ stages is 0-nonblocking. Any MIN that possesses the banyan property (§ 2.4.3) is at least 1-nonblocking. The class of cube-type networks [41, 73, 112] is an example. The k -nonblockingness of some existing networks is studied in § 5.2.2. Obviously, N -nonblocking networks correspond to conventional nonblocking networks.

The general notion of k -nonblockingness can conveniently quantify the power of a MIN, especially when it is under unspecified 1-to-1 traffic (e.g., Theorem 6.7).

5.2.2 k -Nonblockingness of Some Existing Networks

Three theorems about the k -nonblockingness of existing MINs are proven below for later reference and comparison. (s, d) denotes a request from input s to output d .

Theorem 5.1 *Beneš' rearrangeable network is strictly 2-nonblocking.*

Proof: Beneš' network can be obtained by cascading a baseline network (§ 2.4.3) with an inverse baseline, and then removing one of the 2 middle stages (Figure 5.1). However, it is easy to verify that we can keep both middle stages without affecting the functionalities of the network. We focus on the links that interconnect the 2 halves. Links are numbered downwards starting from 0.

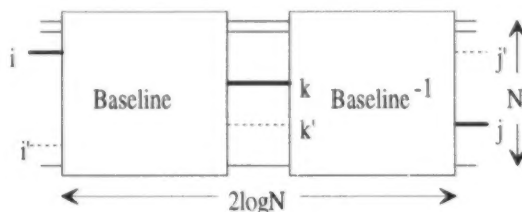


Figure 5.1: Beneš' network. Existing request: (i, j) . New request: (i', j') .

When the network is idle, the first incoming request can always be satisfied. Suppose there exists a connection from input i to output j , using link k at the center. The second request from i' to j' needs a free link $k' \neq k$ at the center. Due to the properties of the baseline, input i' should be able to access at least $\frac{N}{2}$ free links at the center. The worst case happens when i' and i share a switch at the left-most stage. (Each input switch of the baseline can be viewed as the root of a $(\lg N)$ -level complete binary tree, if the switches are treated as nodes. Any output is accessible from the root via a unique path down the tree. A conflict at the root leaves one half of the outputs inaccessible.) Similarly, output j' should be reachable from at least $\frac{N}{2}$ free links at the center. Since there are only $N-1$ free links at the center, the $\frac{N}{2}$

links accessible from i' must have at least one link in common with the $\frac{N}{2}$ links that can reach j' . This common link can be selected as k' . Hence, a request can always be satisfied without disturbing the established connection, if there is one. The network is therefore strictly 2-nonblocking. It is not strictly 3-nonblocking because the situation shown in Figure 5.2 blocks the next request (6, 1). $\square$

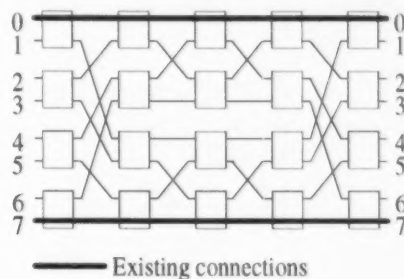


Figure 5.2: The third request (6, 1) is blocked in Beneš network.

Theorem 5.2 *The extra-stage-cube (ESC) network is strictly 2-nonblocking.*

Proof: Clearly, the ESC (§ 2.4.3) is at least 1-nonblocking. Assume there exists a connection when the second request arrives. Lemma 3.1 indicates that switches in stages $1, 2, \dots, n-1$ can be classified into path-0 and path-1 switches. If the existing request is using path 0 (1), then path 1 (0) can always satisfy the second request. Hence, the network is strictly 2-nonblocking. A counter-example similar to that in Theorem 5.1 can be constructed to prove that it is not strictly 3-nonblocking. $\square$

Note that adding another SE stage to the input side of the ESC, giving a 2-GCN (Definition 3.1) that provides 4 alternative paths between every input/output pair, still gives a strictly 2-nonblocking instead of a strictly 4-nonblocking MIN.

We now examine the $v(m, n, r)$ Clos network (§ 2.4.1). Let $\min(a, b, \dots)$ be the minimum of $a, b, \dots$, and let $s(i)$ be the switch with which terminal i is associated.

Theorem 5.3 *$v(m, n, r)$ is m -nonblocking in the strict sense for $1 \leq m < 2n-1$.*

Proof: Because there are m switches in the middle stage, m alternative paths are available for every input/output pair. A request is blocked only when all its m paths

are occupied. Each existing connection can only eliminate at most one of an incoming request (i, j) 's alternative paths. Therefore, $v(m, n, r)$ is at least m -nonblocking in the strict sense. It suffices to show that the $(m + 1)$ th request can be blocked to establish that the network is not $(m + 1)$ -nonblocking in the strict sense.

Label switches in each stage downwards starting from 0. Evidently, an existing connection (g, h) eliminates one of (i, j) 's path if $s(g) = s(i)$, or $s(h) = s(j)$, or both. Let there be $l = \min(n - 1, m)$ existing connections using $s(j)$. Consider 2 cases:

$m \leq n - 1$: All m alternative paths have been eliminated so i cannot reach j .

$m > n - 1$: All m paths leading to $s(j)$ may become unavailable to (i, j) if there are $m - l$ existing connections using $s(i)$. Since $0 < m - l < n$, it is always possible to generate these $m - l$ connections to block the $(m + 1)$ th request. $\square$

Theorem 5.3, however, is not a good scheme for constructing k -nonblocking MINs because the Clos network suddenly becomes fully nonblocking in the strict sense when m reaches $2n - 1$ (e.g., a typical choice $n = 2$ only gives 1-, 2-, and N -nonblocking MINs). Choosing $r = 2$ instead of $n = 2$ solves this "discontinuity" problem but the resulting network is not cost-effective due to the use of large switches in both stages 1 and 3. This motivates the construction scheme given below.

5.3 Constructing a k -Nonblocking Network

This section shows how a class of strictly k -nonblocking networks, where $k \geq 2$, can be obtained from Beneš' rearrangeable network. Existing nonblocking networks, such as the one constructed in [79] (as a variant of Cantor's network [23]), generally have a large number of methodically selected *inactive* terminals (i.e., no connection involves them). This may simplify routing and guarantee nonblocking operation (by limiting the workload) simultaneously. In contrast, we allow more active terminals while nonblocking operation is guaranteed for workload less than k .

We first define a class of networks $C(M, N)$, where N is the size and M is a

parameter. Then, the relationship between the k -nonblockingness of $C(M, N)$ and the parameter M is derived.

$C(M, N)$ is a Clos-like, 3-layer network. There is a column of N 1-to- M demultiplexers in the input layer and a column of N M -to-1 multiplexers in the output layer. In the middle layer is a column of M $N \times N$ Beneš networks. In addition, the network has the following constraint for all $i \in \{1, \dots, N\}$.

C1: The i th input (output) of each Beneš network in the middle layer is connected to the i th demultiplexer (multiplexer) in the input (output) layer.

A $C(4, 8)$ is shown in Figure 5.3. The stages in each Beneš network are labeled from 1 on the left to $2 \lg N - 1$ on the right. The structure of $C(M, N)$ is not regular due to the nature of layers 1 and 3. But when $M = 2^m$, each demultiplexer (multiplexer) can be replaced by an inverse baseline (baseline) network of $\frac{M}{2} \lg M$ 2×2 switches. Only 1 arbitrary chosen input (output) of each baseline (inverse baseline) is kept active. The resulting network is denoted by $C^*(M, N)$. Using existing results in the literature, it can be shown that $C^*(M, N)$, counting all active/inactive terminals, is topologically equivalent to a Beneš rearrangeable network of size NM .² So the following results on $C(M, N)$ applies to Beneš' network directly.

Before we proceed, basic properties of Beneš' network are reviewed. An N -input Beneš network has $2 \lg N - 1$ stages (§ 2.4.4). There is a unique path running between every input and every switch in the middle stage (i.e., stage $\lg N$). Analogously, a unique path runs between every switch in the middle stage and every output. Every switch in the middle stage represents a path from any given input i to output j . Hence, there are $\frac{NM}{2}$ alternative paths for every input/output pair in $C(M, N)$.

We will prove the nonblockingness of $C(M, N)$ in Theorem 5.5 by counting the number of usable switches in the central stage. So let B_b , where $0 \leq b < N$, be the maximum number of middle-stage switches (in all Beneš networks in layer 2) made

²In fact, our original derivation was based entirely on Beneš' rearrangeable network. $C(M, N)$ was introduced to simplify the proof.

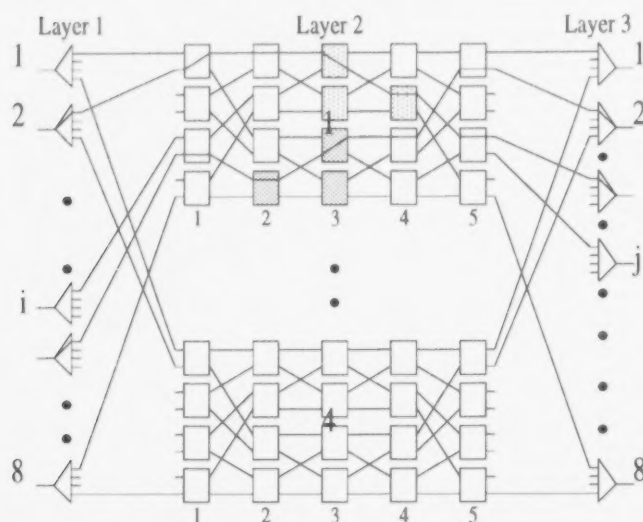


Figure 5.3: Highlighted switches in $C(4, 8)$ are unusable by any path from i to j .

inaccessible to any pair of idle input i and output j when there are b busy connections. Obviously, $B_0 = 0$. In general:

Lemma 5.4 $B_b = \underbrace{\left(\frac{1}{2} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{8} + \cdots\right)}_{b \text{ terms}} \frac{N}{2}, \text{ for } 1 \leq b < N.$

Proof: Given a request involving idle input i and output j , the number of middle-stage switches made inaccessible to the request (i, j) by an existing connection, due to the sharing of a switch (with (i, j)) in stage s is $\frac{N}{2^{s+1}}$, where $1 \leq s \leq \lg N$. By symmetry, the number of middle-stage switches made inaccessible to the request due to the sharing of a switch in stage $2\lg N - s$ is also $\frac{N}{2^{s+1}}$. See Figure 5.3 for an example. Therefore, $B_1 = \frac{N}{4}$, corresponding to the sharing of a switch in either stage 1 or $2\lg N - 1$. (Note that once a worst case conflict has occurred on one side of the network, any conflict on the other side of the network only affects middle-stage switches already made inaccessible by the worst case conflict. Therefore, $B_1 \neq \frac{N}{4} + \frac{N}{4}$.) Similarly, $B_2 = B_1 + \frac{N}{4} = \frac{N}{2}$, corresponding to the sharing of switches in stages 1 and $2\lg N - 1$ (but not necessary within the same Beneš network in layer 2).

Constraint **C1** guarantees that at most 2^{s-1} existing connections can share a switch in stage s (or $2\lg N - s$) with any path running between i and j , where

$1 \leq s \leq \lg N$. Therefore, $B_3 = B_2 + \frac{N}{2^3} = (\frac{1}{2} + \frac{1}{2} + \frac{1}{4})\frac{N}{2}$, corresponding to the sharing of 2 switches in stages 1 and $2 \lg N - 1$, plus 1 switch in either stage 2 or $2 \lg N - 2$. By continuing in this direction, we obtained the general expression for B_b . $\square$

Remark: the counting argument used in Lemma 5.4 is similar in principle to Cantor's argument [54] with the notable exception that we restrict the workload in the network to at most k calls.

Given $M = 2^m$, the highest degree of k -nonblockingness of $C(M, N)$ can be computed by Theorem 5.5, which is the main result in this section.

Theorem 5.5 $C(2^m, N)$ is strictly $\min(N, 2(2^{2^m} - 1))$ -nonblocking.

Proof: It is evident from Lemma 5.4 that B_b increases monotonically with b . $C(M, N)$ is k -nonblocking if one or more of the $\frac{MN}{2}$ middle-stage switches are accessible to every pair of idle input i and output j whenever there are $< k$ existing connections. Formally, we have $B_{k-1} < \frac{NM}{2}$, which reduces to:

$$\underbrace{\left(\overbrace{\frac{1}{2} + \frac{1}{2}}^2 + \overbrace{\frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4}}^4 + \overbrace{\frac{1}{8} + \dots + \dots}^8 + \dots \right)}_{k-1 \text{ terms}} < 2^m \quad (5.1)$$

due to Lemma 5.4. Let us introduce an integer variable:

$$\beta = \frac{1}{2^1} \cdot 2^1 + \frac{1}{2^2} \cdot 2^2 + \frac{1}{2^3} \cdot 2^3 + \dots + \frac{1}{2^\beta} \cdot 2^\beta.$$

Since there are $k - 1$ terms on the left side of inequality 5.1 and 2^m is an integer, the largest possible k , called $k_{\max}$, makes the left side an integer and satisfies:

$$\begin{aligned} 2^1 + 2^2 + \dots + 2^{\beta-1} + (2^\beta - 1) &= k_{\max} - 1 \\ 2^1 + 2^2 + \dots + 2^{\beta-1} + 2^\beta &= k_{\max} \\ \beta &= \lg\left(\frac{k_{\max}}{2} + 1\right). \end{aligned}$$

Inequality 5.1 can be rewritten as $\beta - \frac{1}{2^\beta} < 2^m$, which means $\beta \leq 2^m$. So:

$$\begin{aligned} \lg\left(\frac{k_{\max}}{2} + 1\right) &= 2^m \\ k_{\max} &= 2(2^{2^m} - 1). \end{aligned} \quad (5.2)$$

Since $k_{\max}$ is bounded by N , we conclude that $C(2^m, N)$ is k -nonblocking in the strict sense, where $k \leq \min(N, 2(2^{2^m} - 1))$, because no path-selection algorithm is required to satisfy a request whenever there are $< k_{\max}$ existing connections. $\square$

Remarkably, Equation 5.2 is independent of N . We will omit the subscript in $k_{\max}$ whenever it is understood from the context.

5.3.1 Interpretation and Application

If an $(\frac{N}{2})$ -nonblocking network, where $N = 2^n$, is needed, then substituting $k = 2^{n-1}$ into Theorem 5.5 yields:

$$2^{n-1} \leq 2(2^{2^m} - 1).$$

Since we are dealing with integers, we can say:

$$2^{n-2} < 2^{2^m}$$

$$n - 2 < 2^m$$

$$n - 1 \leq 2^m.$$

So $m \geq \lg(n - 1)$ is sufficient, for $n > 2$. Table 5.1 shows the first few values of m and the corresponding $k_{\max}$. When $m = 0$, we have a Beneš network of size N with all terminals being active, for which $k_{\max}$ is 2. It precisely agrees with Theorem 5.1.

m	$k_{\max}$
0	2
1	6
2	30
3	510
4	131,070

Table 5.1: m verses $k_{\max}$.

Table 5.1 suggests that, in the proposed construction, a slight increase in m causes $k_{\max}$ to rise sharply. For instance, to build a MIN with $k = 7$, $m = 2$ is needed which actually gives a 30-nonblocking MIN. Although the straight-forward approach illustrated below by example is not cost-effective, it demonstrates that any precisely k -nonblocking MIN can be explicitly constructed.

To build a 3-nonblocking MIN of size N , we can connect in parallel one Beneš and one baseline network, both of size N . A new stage of N demultiplexers (multiplexers) are connected to the input (output) side to allow any input (output) to access either network. The resulting network (Figure 5.4) is exactly 3-nonblocking because 2 requests can use Beneš network while 1 request can be directed to the baseline.

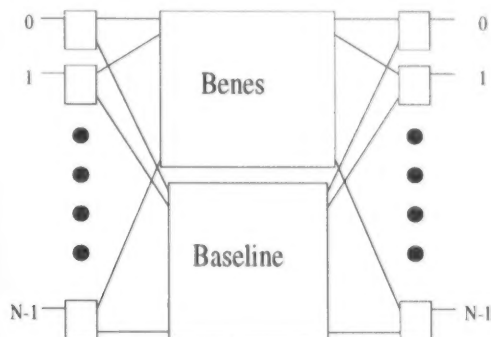


Figure 5.4: Constructing an exactly 3-nonblocking network of size N .

5.3.2 A Special Case: N -Nonblocking

The case $k = N = 2^n$ is examined here because $C^*(M, 2^n)$ can be compared with existing nonblocking MINs. We first compute in Corollary 5.1 the m for an N -nonblocking $C^*(2^m, N)$. Our result essentially coincides with Cantor's result [54].

Corollary 5.1 *When $m \geq \lg \lg N$, $C^*(2^m, N)$ is strictly nonblocking, where $N \geq 4$.*

Proof: By substituting $k = N$ into Theorem 5.5 we get:

$$2^{n-1} < 2^{2^m}$$

$$n - 1 < 2^m$$

$$m \geq \lg \lg N. \quad \square$$

Therefore, we can construct a conventional strictly nonblocking MIN of size N by retaining any one of every $\lg N$ terminals of a Beneš network of size $N \lg N$. Lin and Pippenger's nonblocking network [79] has a parameter similar to m . But in order to

prove the nonblockingness and sustain an acceptable routing time, they strategically set m to $\lfloor \lg 8n \rfloor$. Thus, the proposed construction produces cheaper networks.

5.4 Routing Algorithms

It remains to show that the network constructed above can be controlled efficiently. We devised 2 routing algorithms³ for $C^*(M, N)$: MPL and FirstFit. The former is a probabilistic parallel algorithm while the latter is a deterministic sequential algorithm.

5.4.1 MLP

Idea: MLP (Modified Lin-and-Pippenger) is a direct adoption of the probabilistic algorithm in [79] (will be reviewed below), except that it is applied only when there are $< k$ existing connections (otherwise, it may not terminate).

Data structure: Requires $O(N(\log N)^2)$ space as shown in [79].

Input: A batch of t connection requests. Each request is an ordered pair (i, j) .

Output: Route assignment for the t requests.

Algorithm: Starting from the given input and output terminals for each request, make random routing decisions at the switches so that the left and right halves of a path (the left half of a path cuts through layer 1 and the first half of a Beneš network in layer 2) are produced. If the probability that the left (right) half of the path is free is $> \frac{3}{4}$, then the probability that the full path is free is $\frac{3}{4} \times \frac{3}{4} = \frac{9}{16} > \frac{1}{2}$. At such a high success rate, a trial-and-error approach is effective. After a number of *rounds*, most of the t requests should have been satisfied. See [79] for details.

Analysis: For simplicity, assume $C^*(2^m, N)$ is 2^p -nonblocking, where $0 < p \leq n$. From Theorem 5.5, we need $m \geq \lg p$. The number of links leaving the middle

³A third deterministic algorithm called Parallel DFS, which takes $O(N)$ time, is not covered.

stage (towards the outputs) is 2^{m+n} . When the network is idle, all of them are accessible from every input. The number of inaccessible links in the worst case (i.e., when $k-1$ established connections are present) is $p2^{n-1}$. Thus, the number of accessible links leaving the middle stage is:

$$2^{m+n} - p2^{n-1} = 2^{n-1}(2 \cdot 2^m - p) = 2^{n-1}(2p - p) = 2^{n-1}p.$$

The chance of hitting a free path when trying randomly from either the input or output side is:

$$\frac{2^{n-1}p}{2^{m+n}} = \frac{p}{2^{m+1}} = \frac{1}{2},$$

which is worse than the $\frac{3}{4}$ in [79]. Thus, random search takes a longer time (comparing with that in [79]) to find a free path because $C^*(M, N)$ has fewer alternative paths as a result of allowing a higher ratio of active terminals. However, it is worthwhile to relax m slightly to keep the ratio of free paths $> \frac{3}{4}$. In doing this, both of the probabilistic and deterministic algorithms in [79] will serve $C^*(M, N)$ at their original near-optimal speeds, $O((\log N)^2)$ and $O((\log N)^5)$, respectively. The condition is:

$$\frac{2^{m+n} - p2^{n-1}}{2^{m+n}} \geq \frac{3}{4},$$

which simplifies to $m \geq 1 + \lg p$. Therefore, by increasing m by 1, all algorithms in [79] will work on $C^*(M, N)$ at the above mentioned speeds. To achieve these speeds, an N -nonblocking network needs $m \geq \lg 2n$ which is still smaller than the $m = \lfloor \lg 8n \rfloor$ in [79]. Hence, we actually obtained a slightly cheaper network without sacrificing routing time (n.b., their network is already "weakly optimal" [79] so large saving is not possible).

5.4.2 FirstFit

Idea: Try to search for a usable switch c in the middle stage and then verify that the two path segments from input i to c , and from c to output j , are free.

Data structure: Use 3 bits to encode the state of every switch (Figure 5.5) not in the first/last m stages. $O(N(\log N)^2)$ space is needed, which equals that in [79].⁴

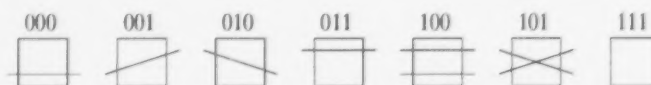


Figure 5.5: Three bit encoding of the state of a switch.

Input: A connection request (i, j) .

Output: Middle-stage switch c which specifies the path uniquely.

Algorithm: Function headers and main program are presented in pseudo code:

```

int  mstate(c);      /* returns the status of middle-stage switch  $c$  */
bool leftfree( $i, c$ ); /* returns true if all switches in the unique path between
                        input  $i$  and middle-stage switch  $c$  allow the connection, otherwise false */
bool rightfree( $c, j$ ); /* returns true if all switches in the unique path between
                        middle-stage switch  $c$  and output  $j$  allow the connection, otherwise false */
void foundpath( $c, i, j$ ); /* updates the data structure to reflect the path assignment
                        defined by input  $i$ , middle-stage switch  $c$ , and output  $j$  */
int  MSB( $i$ );        /* returns the most significant bit of  $i$  */

int function FirstFit( $i, j$ ) /* assume C-like semantics for operators */
1  for  $c = 0$  to  $2^{m+n-1} - 1$  step 1 do /* loop for all middle-stage switches */
2      if ((mstate( $c$ ) == 7) or (((MSB( $i$ ) << 1) | MSB( $j$ )) == mstate( $c$ ))) then
        /*  $c$  is usable if it is idle or it's state is compatible with the request */
3          if (leftfree( $i, c$ ) and rightfree( $c, j$ )) then
4              foundpath( $c, i, j$ )
5              return( $c$ )
        endif
    endif
endfor
6  return(-1) /* no free routes, probably because workload >  $k$  */
endfunc

```

Analysis: `mstate()` and `MSB()` clearly take $O(1)$ time. Each of `leftfree()` and `rightfree()` checks the status of $O(\log N)$ switches so they take $O(\log N)$ time. Similarly, `foundpath()` takes $O(\log N)$ time. Hence, every iteration of the

⁴Their deterministic algorithm requires more space than ours due to a large constant factor.

main loop in FirstFit takes $O(\log N)$ time. The complexity of FirstFit is thus $O(2^{m+n} \log N)$. However, c in line 1 can be incremented in a better way: whenever `leftfree()` or `rightfree()` reports a conflict with an existing connection when they scan the network from outside towards the center, all other middle-stage switches accessible from the switch output at which the conflict occurs can be skipped. Now the main loop iterates at most k times so the complexity reduces to $O(k \log N)$. If k is a constant, FirstFit takes only $O(\log N)$ time. If k is $O(N)$ then the complexity becomes $O(N \log N)$. When k is small, FirstFit could be faster than the $O((\log N)^5)$ time of the deterministic MLP [79].

5.5 Removing Dummy Switches

When $m > 1$, some unused switches can be removed from $C^*(M, N)$. The number of removable switches, and the precise cost of the network is given below:

Corollary 5.2 *The number of removable switches in $C^*(2^m, 2^n)$, where $m > 1$, is $(m-2)2^{n+m} + 2^{n+1}$.*

Proof: There are 2^n active terminals on each side of $C^*(2^m, 2^n)$. The number of stages on the input side that can have switches removed is easily seen to be $m-1$. If the stages (in all layers) are numbered from left to right, starting from 1, then there are 2^{m-1} switches for every active input in stage 1; so $2^n(2^{m-1} - 2^0)$ switches in stage 1 are removable. 2^0 is the number of switches accessible from the active input and hence cannot be removed. In general, the total number of removable switches is:

$$2[2^n \sum_{s=1}^{m-1} (2^{m-1} - 2^{s-1})] = (m-2)2^{n+m} + 2^{n+1}. \quad \square$$

The actual cost of the network measured by the number of switches is:

$$[2(n+m)-1]2^{n+m-1} - [(m-2)2^{n+m} + 2^{n+1}] = (2n+3)2^{n+m-1} - 2^{n+1}.$$

When $2^m = n$, which results in an N -nonblocking network, the cost is $2^{n-1}(2n^2 + 3n-4)$; and that is still $O(N(\log N)^2)$.

5.6 Conclusion

We accomplished 3 tasks in this chapter. First, the general notion of k -nonblocking networks provides a conceptual framework for achieving nonblocking operation at lower cost. Rearrangement/blocking can be eliminated by choosing a large k (for MIMD machines) or by dividing each communication step that involves $> k$ connections into phases (for SIMD machines). The idea applies to both wide- and strict-sense nonblocking MINs. We also studied the k -nonblockingness of some existing MINs.

Second, a systematic scheme for constructing strictly k -nonblocking MINs, based on Beneš' network, is given. If one out of every 2^m terminals of a Beneš network is kept active, then a suitable m can be computed to guarantee k -nonblockingness. We allow the highest ratio of active terminals while k limits the workload to sustain nonblocking operation. Hence, computation tasks should not be distributed if extensive communication is constantly needed. Note that other approaches to constructing k -nonblocking networks exist.

Third, efficient routing algorithms were devised. Since the parallel algorithm MLP is near optimal, we actually obtained a nonblocking network slightly cheaper than that in [79] without sacrificing routing time. Thus, parameterizing nonblocking networks by k does not necessarily incur cost/routing overheads. FirstFit is a good sequential algorithm for small or fixed k 's.

One open problem is the construction of wide-sense k -nonblocking MINs. (We already found that a $v(m, n, r)$ Clos network is wide-sense m -nonblocking for $0 < m < n + \min(n, r) - 1$ [69].) Another interesting direction arises from a review of some MINs we considered. Table 5.2 shows the minimum depth required for any MIN to be k -nonblocking when the width (§ 2.1) is kept at N . The '?' entries $k = 3, 4, \dots, N$, correspond to open problems. Chapter 6 solves some of them.

A second interesting direction is the design of low cost MINs to achieve nonblocking operation for prescribed traffic patterns. For example, it is not difficult to prove that the rearrangeable network in [60, Figure 10] is wide-sense nonblocking under first-in-last-out traffic (i.e., the most recent request is released first) [69]. It is not

k -nonblockingness	Minimum depth	Example
0	≥ 1 (but $< \lg N$)	SE (§ 5.2.1)
1	$\geq \lg N$	baseline (§ 5.2.1)
2	$\geq \lg N + 1$	ESC (Theorem 5.2)
3	?	?
$\vdots$	$\vdots$	$\vdots$
N	?	?

Table 5.2: Minimum depth necessary to be k -nonblocking.

known whether it is wide-sense nonblocking under first-in-first-out traffic too.

Chapter 6

Width N Wide-Sense Nonblocking Networks

This chapter mainly discusses the synthesis of nonblocking networks made of square switches (where the width of a size N network is also N). We first show in § 6.2 that no width N MIN can be strictly nonblocking. In § 6.3, a width N multi-path recursive network called the *Quadruplet* (Q) is introduced. Important properties of the Q network are given in § 6.4. We prove that Q is wide-sense nonblocking in § 6.5. More information about Q is presented in § 6.6. By transforming Q into a proper MIN in § 6.7, we uncover a fundamental difference between wide- and strict-sense nonblocking MINs: the former can have width N but the latter cannot. § 6.8 proves that a wide-sense nonblocking MIN can be built by serially cascading a number of shuffle/exchange stages. A new implication of the notion of universality is also shown.

6.1 Introduction

Many existing MINs are made of small, square switches. For example, the 2x2 crossbar is the major building block in [41] and [111]. The 3x3 crossbar is used in the Gamma network [123] while the 4x4 crossbar is used in [50]. Due to their wide-spread use, MINs made of square switches deserve our attention. The width (§ 2.1) of a size

N MIN made of square switches is N because the number of links between any 2 adjacent stages is N . This chapter mainly discusses this kind of MINs. If the width of a network is not N , it is understood that non-square switches have been used.

Our work is motivated by two related open problems. We now discuss the first one. Intuitively, a more powerful MIN can be constructed if a larger width is allowed while the depth is fixed, or a larger depth is allowed while the width is fixed. The former case is well covered in the literature (e.g., [120] has a probabilistic study; Theorems 2.1 and 5.3 for the Clos network form another example) because limiting the depth improves the end-to-end delay of a MIN. The latter case is an interesting area for theoretical studies. If the width is fixed at N , then the minimum depths required to construct networks having various capabilities are given in Table 6.1. The ‘?’ entries correspond to open problems (c.f. the last row in Table 5.2).

Minimum depth	Network obtainable	Example
1	blocking, non-full-access	SE (§ 2.4.2)
$\lg N$	blocking, full-access	Ω (§ 2.4.3)
$2 \lg N - 1$	Rearrangeable	Beneš (§ 2.4.4)
?	wide-sense nonblocking	?
?	strict-sense nonblocking	?

Table 6.1: Minimum depth required for width N MINs of various capabilities.

Based on Beneš’ work, we show in § 6.2 that strictly nonblocking MIN cannot have width N . It is not known whether width N wide-sense nonblocking MINs exist, except an example network of size 4 (Figure 6.9) given by Beneš [14]. This network does not generalize to $N > 4$. We resolve this issue by proving that N is a tight lower bound on the width of wide-sense nonblocking MINs. Although a width N MIN cannot be strictly nonblocking, what is the highest strict-sense k -nonblockingness (Definition 5.1) attainable? This question is addressed too.

We now examine the second motivation. Some definitions are given first. The *universality* of a MIN is the number of passes it requires to perform an arbitrary permutation [110, 118]. For example, the universality of the GCN is 3 (§ 2.4). An *iterated* MIN [43] uses one permutation to interconnect all adjacent stages. The universality

of iterated MINs has received extensive attention because any target permutation can be performed by recirculating it through a MIN with only a few stages. Table 6.2 summarizes some important results in this direction. A permutation p is *universal* if a rearrangeable iterated MIN can be built using p . Since the introduction of the perfect shuffle in [117] as the first universal permutation, the question “How many shuffle/exchanges (SEs) are needed to attain rearrangeability?” [124, Chapter 4] has interested many researchers. The current confirmed¹ upper and lower bounds are $3n-4$ and $2n-1$, respectively, where $n = \lg N$. The set of universal permutations has been characterized in [43] and all elements in the set are equivalent (i.e., if x stages of SE is rearrangeable, so is x stages of any other universal permutation).

Researchers	Year	Achievements
Stone [117]	'71	n^2 stages of SE is rearrangeable, algorithm given
Parker [95]	'80	$3n$ stages of SE is rearrangeable, no algorithm
Wu and Feng [129]	'81	$3n-1$ stages of SE is rearrangeable, algorithm given
Kothari et. al. [66]	'83	$3n-3$ stages of SE is rearrangeable ($N = 16$ or 32)
Huang and Tripathi [53]	'86	$2n-1$ stages of SE are necessary, $3n-3$ stages are sufficient for rearrangeability
Raghavendra and Varma [103]	'86	$3n-4$ stages of SE are sufficient for rearrangeability, algorithm given for a 5-stage SE with $N = 8$
Linial and Tarsi [80]	'89	$3n-4$ stages of SE is rearrangeable, algorithm given
Cam and Fortes [22]	'90	$2n-1$ stages of SE are both necessary and sufficient for rearrangeability, no algorithm
Fiduccia and Jacobson [43]	'91	Characterized the set of universal permutations
Kim and Raghavendra [63]	'91	A better algorithm for a 5-stage SE with $N = 8$
Feng and Seo [42]	'94	$2n$ stages of SE is rearrangeable, algorithm given

Table 6.2: Some results in the study of iterated MINs.

Now, it becomes natural to ask “How many SEs are needed to attain nonblockingness?” Hwang [55] has proven that a strictly nonblocking MIN cannot be obtained by serially cascading SE stages. However, the case for wide-sense nonblockingness is an open problem. Note that one can declare a MIN as rearrangeable without giving

¹In 1994, Varma and Raghavendra [124, Chapter 4] claimed (without explanation) that the proof in [22] (see Table 6.2) is incorrect. They also pointed out the mistake in Sovis's attempt [116] to prove that $2n-1$ stages of SE are sufficient for rearrangeability. To exercise caution, we are not using Feng and Seo's [42] new result (see Table 6.2) here. But once the correctness of [42] is confirmed, our result in Corollary 6.2 can easily be updated by changing all $3n-4$ to $2n$.

a routing algorithm (exhaustive search can always find a solution). But to declare a MIN as wide-sense nonblocking, one must give, or at least prove the existence of, a routing algorithm that can satisfy any valid request/release sequence.² This is in general a very difficult task. For example, Beneš exhaustively enumerates all possible states of his 4×4 network [14]. This method suffers from “state explosion” when N is large.

We have covered our two motivations. To avoid state explosion, we designed a recursive network, called the *Quadruplet* (Q), as the vehicle for expressing the idea of our routing algorithm (traditionally, routing algorithms are designed for existing networks). Hence, Q is wide-sense nonblocking. By transforming Q into a proper MIN of width N , a fundamental difference between wide- and strict-sense nonblocking MINs not obvious from their definitions is identified: the former can have width N while the latter cannot ([38] gives another difference concerning the cost). By substituting sub-networks in Q with SE stages, we derive an upper bound on the number of SE stages sufficient for wide-sense nonblockingness. Both open problems that motivated us are thus solved.

The major task in this chapter is to show that Q is a wide-sense nonblocking network. Properties of the Q network are highlighted. Beside solving the two open problems, we also extend the primary result to conclude that:

1. Any width N MIN can at most be strictly $(N-2)$ -nonblocking.
2. By serially cascading a certain number of any rearrangeable MINs of width N , a wide-sense nonblocking MIN can be obtained.
3. The set of (universal) permutations that can produce rearrangeable iterated MINs is identical to the set of permutations that can produce wide-sense nonblocking iterated MINs.

²A wide-sense nonblocking network has *control-dependent-nonblocking capability* [131].

6.2 Width of Strictly Nonblocking MINs

This section shows that the width of a strictly nonblocking MIN (not necessary made of square switches) of size N must be $> N$. Theorem 6.1 is the main conclusion of Beneš' 1981 paper on wide-sense nonblocking MINs made of square switches. *Busy parallel paths* (BPPs) are paths that are busy, and they do not share any switch.

Theorem 6.1 [14]: *Any network made of square switches is wide-sense nonblocking if and only if there exists a routing algorithm that makes all network states that contain BPPs unreachable.*

Theorem 6.1 allows us to claim that:

Corollary 6.1 *Any strict-sense nonblocking MIN (not necessary made of square switches) of size N must have width $> N$.*

Proof: Note that parallel paths exist in all width N MINs that do not have an $N \times N$ switch. By definition, a strict-sense nonblocking MIN allows any available path to be chosen randomly. Hence, it is easy to generate a pair of BPPs in such networks (an example that illustrates the idea is given below). The existence of BPPs implies that a blocking state is reachable [14]. Therefore, a MIN of width N cannot be strict-sense nonblocking. Obviously, no MINs of width $< N$ can be strict-sense nonblocking. In conclusion, strict-sense nonblocking MIN of size N must have width $> N$. $\square$

In Beneš' network (§ 2.4.4), if the top-most links are randomly assigned to a request from input 0 to output 0, and the bottom-most links are randomly assigned to a request from input $N-1$ to output $N-1$, a pair of BPPs is formed (Figure 5.2).

6.3 Definitions and Notations

We focus on circuit-switching operation with 1-to-1 requests only. A connection request is abbreviated by (a, b) , where a and b are idle input and output terminals, respectively. The state of a 2×2 switch is either *swap* or *straight* (§ 2.1). The *state*

of a network is composed of the states of all the switches. In diagrams, an $N \times N$ switch/sub-network is marked ' N '. We define the *Quadruplet* (Q) network recursively:

Definition 6.1 A size 2 *Quadruplet* network, denoted by Q_2 , is a 2×2 crossbar switch.

Definition 6.2 A Q_N , where $N \geq 3$, is a network with N inputs and N outputs made of 4 Q_{N-1} 's, as illustrated in Figure 6.1.

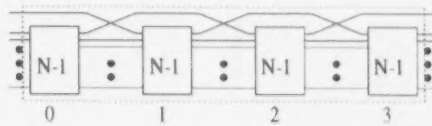


Figure 6.1: Q_N .

Because of the link between stages 0 and 2 (or 1 and 3), Q_N is not considered a proper MIN. The width of Q_N is N . Switches/stages are numbered left-to-right starting from 0. In fact, every $(N-1) \times (N-1)$ sub-network in the Q_N network can be any switch/sub-network that is nonblocking. The inputs/outputs of the sub-networks are indistinguishable and hence their labeling order is immaterial. Figure 6.2 depicts the structure of the first quarter of an expanded Q_5 .

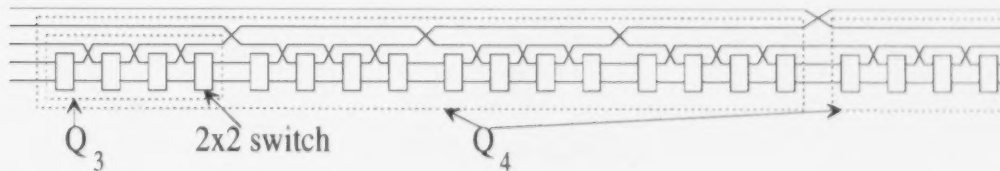


Figure 6.2: The first quarter of a Q_5 .

We now describe our routing model. The current state of a width N network decides what permutation it is implementing. If the current permutation cannot accommodate an incoming request, the states of some idle switches are changed such that the network implements a new permutation which can accommodate the request. Note that busy paths are never disturbed if the network is nonblocking. Links along the used path are marked as busy. The input to the routing algorithm is either a

connection *request* or a *release*. For a request, the algorithm generates as output the path assigned to the request. The status of the network is updated accordingly. For a release, the network status is updated with no output generated. A *path* is uniquely specified by the sequence of switches it uses.

6.4 Basic Properties of the Q Network

Three lemmas in this section highlight properties of the Q network used by the main theorem in next section. The inputs (outputs) of Q_N can be classified into 2 types: the single input a (output a') and the set of indistinguishable inputs B (output B') (Figure 6.3a). All outputs in B' are equivalent in the sense that given any input t , the same set of alternative paths are available for any request (t, b') , where $b' \in B'$. Similarly, all inputs in B are equivalent.

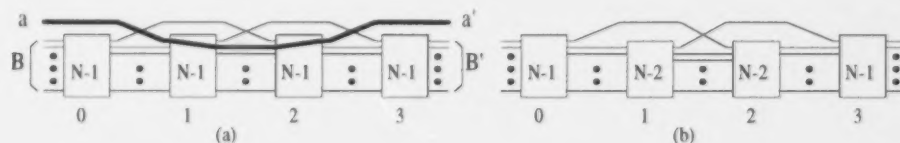


Figure 6.3: a) Terminal labeling of a Q_N , b) Busy path (a, a') removed from graph.

Lemma 6.2 *After serving an arbitrary connection request, Q_N reduces to a network with at least one embedded Q_{N-1} , where $N > 3$.*

Proof: Connection requests for Q_N can be classified into 4 types:

(a, a') : The only feasible path involves switches 1 and 2 (Figure 6.3a). After the connection is established, the rest of the network is depicted in Figure 6.3b.

(a, b') : There are 2 ways to realize this type of request. The first path uses switches 1 and 3 (Figure 6.4a). The second path uses switches 1, 2, and 3 (Figure 6.4b).

(b, a') : Symmetrical to the (a, b') case. Two possible paths exist.

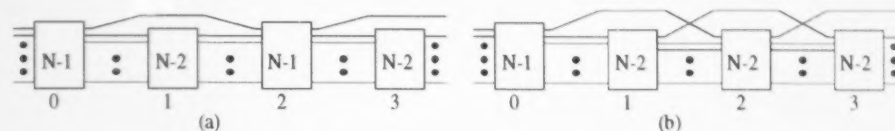


Figure 6.4: Network graphs resulting from the 2 ways to realize (a, b') .

(b, b') : There are 3 possible paths. Path A involves switches 0, 1, and 3 (Figure 6.5a). Path B, which is a mirror image of path A, involves switches 0, 2, and 3. Path C involves switches 0, 1, 2, and 3 (Figure 6.5b).

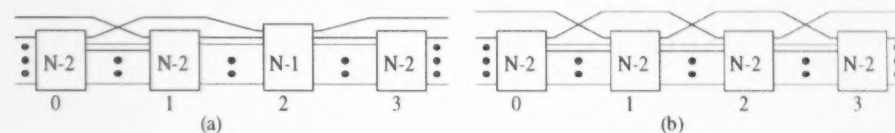


Figure 6.5: Network graphs resulting from the 2 ways to realize (b, b') .

Clearly, whenever ≤ 3 sub-networks are used to satisfy the request, the resulting network has at least 1 free $(N-1) \times (N-1)$ sub-network (i.e., Q_{N-1}). Only path C in the (b, b') case uses all 4 sub-networks in Q_N . Yet, the network in Figure 6.5b is exactly a Q_{N-1} . Therefore, after randomly assigning a free path to an incoming request, Q_N always gives a network with at least 1 embedded Q_{N-1} , where $N > 3$. $\square$

Next lemma supplements Lemma 6.2.

Lemma 6.3 *After serving an arbitrary connection request, Q_3 reduces to a network with at least one Q_2 embedded, if path C is not used to route requests of type (b, b') .*

Proof: Similar to the proof of Lemma 6.2, whenever ≤ 3 switches are used in satisfying a request, Q_3 always reduces to a network which has 1 or more Q_2 's. Only path C in case (b, b') uses all 4 switches in Q_3 . A pair of parallel paths is created if it is chosen. So we avoid using path C to route requests of type (b, b') . $\square$

An outline of the algorithm for routing requests on the Q network is given below. The nonblockingness of Q_N hinges on the priority scheme in Table 6.3, of which Lemma 6.3 is a special case. To satisfy a request, the *high priority path* (HPP) is used whenever possible, otherwise, the *low priority path* (LPP) is taken.

Request Type	Switch usage	
	high priority path	low priority path
(a, a')	1,2	N.A.
(a, b')	1,2,3	1,3
(b, a')	0,1,2	0,2
(b, b')	0,1,3 or 0,2,3	0,1,2,3

Table 6.3: Priority assignment for the 4 types of requests.

```

proc QRoute( $Q_N, s, t$ ) /* ( $s, t$ ) is the connection request */
  if ( $N < 3$ ) then /* 2x2 switch, routing is trivial */
    update network status to record busy path ( $s, t$ )
  else
    consult Table 6.3 and select a path  $\mathcal{P}$  for ( $s, t$ )
    for every sub-network  $\mathcal{Q}$  (of type  $Q_{N-1}$ ) in path  $\mathcal{P}$  do
      call QRoute( $\mathcal{Q}, s', t'$ ) /* ( $s', t'$ ) is the request to be satisfied by  $\mathcal{Q}$  */
    endfor
  endif
endproc

```

For example, to route (a, a') , the Q_{N-1} 's in stages 1 and 2 receive requests of types (a, b') and (b, a') , respectively. Therefore, routing is a recursive process with branching factor ≤ 4 .

Any pair of 2 arbitrary requests fall into one of the 16 combinations in Table 6.4. Except for impossible combinations (indicated by '-'), each table entry gives the labels of the switches always shared by the 2 requests, no matter what paths (HPP or LPP) they are using. The '?' case is discussed in Lemma 6.4 (only one '?' case needs to be examined because of symmetry).

	(a, a')	(a, b')	(b, a')	(b, b')
(a, a')	-	-	-	1 and/or 2
(a, b')	-	-	?	3
(b, a')	-	?	-	0
(b, b')	1 and/or 2	3	0	0 and 3

Table 6.4: Switch(es) shared by any 2 requests.

Lemma 6.4 For $N \geq 3$, if Q_N is initially in a safe state³ and all subsequent requests are routed by QRoute, then any 2 busy paths in Q_N share at least 1 switch.

³A state from which no blocking state is reachable under the routing scheme.

Proof: Only the ‘?’ case in Table 6.4 requires our attention. A pair of BPPs is generated if and only if both (a, b') and (b, a') use their LPPs (Figure 6.6a). We show that our priority scheme avoids this situation if Q_N is initialized properly.

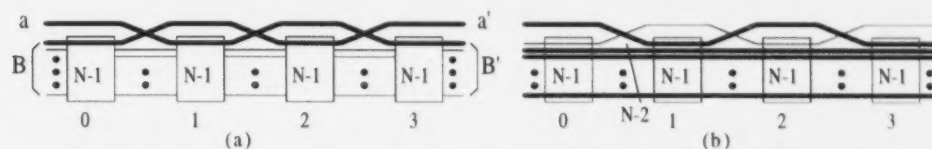


Figure 6.6: a) BPPs (a, b') and (b, a') . b) $N-2$ (b, b') requests using their LPPs.

We examine how the pair of BPPs is formed. Without loss of generality, let (a, b') be an existing connection when the request (b, a') arrives. Since the HPP for (b, a') uses switches 0, 1, and 2, the LPP using switches 0 and 2 is chosen only if either all $N-2$ links between switches 0 and 1 are busy, or all $N-2$ links between switches 1 and 2 are busy, or both. Hence, there must be $N-2$ existing connections of type (b, b') , all using their LPPs (Figure 6.6b), to force (b, a') to use its LPP.

Let (b_i, b'_j) be the most lately arrived request among the $N-2$ requests of type (b, b') . (b_i, b'_j) is forced to take the LPP (switches 0, 1, 2, and 3) only if the direct links between switches 0 and 2, and that between 1 and 3, are both busy. This requires both of the busy paths (a, b') using switches 1 and 3, and (b, a') using switches 0 and 2, to be present. We now see a “circular dependency.” In summary, (b, a') uses its LPP only if all $N-2$ (b, b') connections are using their LPPs. The most recent request (b_i, b'_j) among the (b, b') ’s uses its LPP only if both (a, b') and (b, a') are existing connections and are using their LPPs.

In general, once the system is in a safe state (e.g., an idle network), BPPs cannot be created by the routing scheme. So any 2 busy paths share at least 1 switch. $\square$

Remarks: if the system is initialized to an *unsafe state*,⁴ say, $N-2$ busy connections of type (b, b') all using their LPPs, then there exists request/release sequences, say, request (a, a') or (b, b') , that can force the system into a blocking state. Yet, some

⁴A state from which a blocking state is reachable even if the routing algorithm is in control.

request/release sequences may drive the system from an unsafe state to a safe state. Now, it becomes clear that the ‘?’ in Table 6.4 should be ‘1 and/or 2.’

6.5 The Q Network is Wide-Sense Nonblocking

This section uses Lemmas 6.2, 6.3, and 6.4 to prove that the Q network is wide-sense nonblocking. We require 2 new definitions: Q_1 is a single link (or a series of links) and Q_0 is a null graph. Q_0 , Q_1 , and Q_2 are considered wide-sense nonblocking.

Theorem 6.5 Q_N is a wide-sense nonblocking network, where $N \geq 3$.

Proof: The network state space is simplified to give the transition diagram in Figure 6.7. A transition from Q_i to Q_{i-1} , where $N \geq i \geq 1$, means that on arrival of an arbitrary request, QRoute can satisfy the request and the resulting network has at least one Q_{i-1} embedded. A transition from Q_{i-1} to Q_i means that if one of the $N-i+1$ busy connections is released, the resulting network has at least one Q_i embedded. To conclude that a given physical network Q_N is wide-sense nonblocking, it suffices to show that every transition can be realized by QRoute.

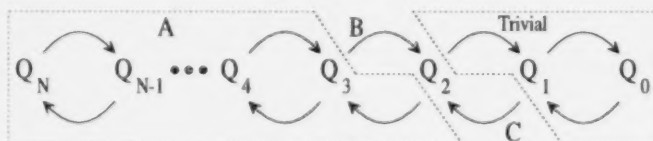


Figure 6.7: State transition diagram for Q_N .

For each $Q_i \rightarrow Q_{i-1}$ transition in area A, where $N \geq i > 3$, Lemma 6.2 guarantees that a path (passing through the Q_i embedded in Q_N) can be found to satisfy an arbitrary request, leaving behind a free Q_{i-1} embedded in Q_N (Lemma 6.2 also ensures that the Q_{i-1} 's used in the path can handle their corresponding requests). Every $Q_{i-1} \rightarrow Q_i$ transition in area A, where $N \geq i > 2$, is realizable because by removing from Q_N , one at a time in arbitrary order, the $N-i$ paths that remains busy after the release, a Q_i can be obtained due to Lemma 6.2. Lemma 6.3 is customized to

handle transition B. Transition C is realizable due to Lemma 6.4 as explained below. Suppose θ is the free path (in Q_1) and τ is the path to be released. Although θ is a free path, θ and τ must share at least one 2×2 switch, which becomes the Q_2 we need once τ is released. Hence, we can always get from Q_1 to Q_2 . All transitions in area 'Trivial' are obviously realizable. Since QRoute can satisfy all valid request/release sequences, Q_N is wide-sense nonblocking. $\square$

Intuitively, Theorem 6.5 holds because Lemma 6.4 provides a strategy for avoiding BPPs while Lemmas 6.2 and 6.3 ensure that the current network contains at least one Q network for the strategy to work on. Alternatively, the proof for Theorem 6.5 may be built on top of Theorem 6.1 and Lemma 6.4, but then Theorem 6.7 ahead will require a separate proof. QRoute has been implemented in C and verified (by simulation experiment) to work correctly for $N < 8$.

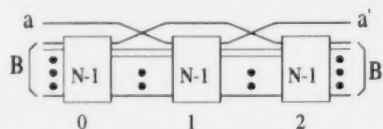
6.6 More About Q_N

This section discusses the simplicity, building block, width, depth, and power of the Q network. Lemma 6.6 hints that the number of stages in Q_N cannot be reduced. Non-exhaustive experimentation shows that Q_N ceases to be nonblocking when smaller switches (e.g., an $(N-2) \times (N-2)$ switch in parallel with a 2×2 switch) are used at any stage. Hence, Q_N seems to be the simplest recursive wide-sense nonblocking network of width N (a recursive rearrangeable network of width N is in [60]).

Lemma 6.6 *The 3-stage network graphically defined in Figure 6.8 is not nonblocking.*

Proof: The following request/release sequence forces the network into a blocking state no matter how the requests are routed: $N-1$ requests of type (b, b') are presented to an idle network. The connection (b_i, b'_j) that occupies the direct link between switches 0 and 2 is released. The next request (a, b'_j) is blocked. $\square$

In general, Q_N can use any $m \times m$ crossbar switches ($m \geq 2$) as building blocks without affecting previous results. That is, Definition 6.1 now defines Q_m to be an $m \times m$ switch. Definition 6.2 remains the same for $N > m$. Although the regularity

Figure 6.8: After removing 1 stage from Q_N .

of Q_N will suffer, Definition 6.1 may also define Q_4 to be Beneš' 4x4 wide-sense nonblocking network [14] (Figure 6.9). Definition 6.2 still defines Q_N for $N > 4$.

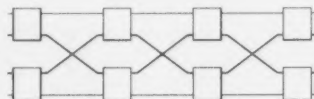


Figure 6.9: Beneš' size 4 wide-sense nonblocking network.

Although Q_N is not meant to be a practical network, it is interesting to know how its depth relates to its size. Let the depth $d_m(N)$ be the number of stages of $m \times m$ switches in Q_N . Definition 6.2 directly implies $d_m(N) = 4d_m(N-1)$. This recurrence equation has a simple solution $d_m(N) = 4^{N-m}$ for the boundary condition $d_m(m) = 1$ (if Beneš' 4x4 network is used as the building block, we have $d_B(N) = 4^{N-3}$ for the boundary condition $d_B(4) = 4$). Note that $d_m(N)$ is also a measure of the cost because there is only one⁵ row of $m \times m$ switches in Q_N .

Recall that a network is k -nonblocking in the strict-sense (§ 5.1) if it appears to be strictly nonblocking whenever there are $< k$ existing connections, where $1 \leq k \leq N$. Although no width N network can be strictly N -nonblocking (Corollary 6.1), Q_N has the highest strict-sense k -nonblockingness possible:

Theorem 6.7 *For $N \geq 3$, Q_N is strictly $(N-2)$ -nonblocking. $N-2$ is a tight upper bound on the strict-sense k -nonblockingness of any network made of square switches.*

Proof: Lemma 6.2 guarantees that all transitions in area A in Figure 6.7 are realizable even if requests are satisfied by randomly assigning free paths to them. Without the simple routing scheme in Lemma 6.3, transition B in Figure 6.7 is still realizable

⁵Two rows of 2x2 switches if Beneš' 4x4 network is used as the blocking block.

although a pair of parallel paths may be generated instead of a Q_2 . Therefore, Q_N is strictly $(N-2)$ -nonblocking. Note that if a width N network is strictly $(N-1)$ -nonblocking, it must also be strictly N -nonblocking because the last pair of idle input/output in a network carrying $N-1$ busy paths must be connected by a free path. Since a network made of square switches cannot be strictly N -nonblocking, it cannot be strictly $(N-1)$ -nonblocking either. So $N-2$ is a tight upper bound on the strict-sense k -nonblockingness of any network made of square switches. $\square$

Notably, Theorem 6.7 implies that Q_N almost becomes a fully nonblocking network in the strict sense as N goes to infinity.

6.7 Transforming Q_N into a Proper MIN

The Q network violates the definition of standard MINs in 2 ways as follows:

1. Non-adjacent stages in Q_N are connected. In a S -stage MIN (we require MINs to be *layered* (§ 2.1)), stage s is connected to stage $s+1$ only, where $1 \leq s < S$.
2. Q_N has multiple links between 2 adjacent Q_{N-1} 's. In a MIN, at most 1 link connects any 2 switches.

We show that by adding dummy switches, Q_N can be transformed into a proper MIN. The transformation technique in Lemma 6.8 is required by subsequent theorems. If $N \neq km$ in Lemma 6.8, we can set $N' = \lceil N/m \rceil m$ and disable $N' - N$ terminals at the top of $Q_{N'}$. But now the width of the network is N' instead of N .

Lemma 6.8 *For $N = km$, Q_N can be transformed into a proper MIN of depth $d_m(N) = 2 \cdot 4^{N-m} - 1$, where m is the switch size and $k \geq 1$ is an integer.*

Proof: Two transformations remove the 2 violations in sequence:

1. To remove the first violation we apply a recursive procedure. Assume Q_m is a valid MIN and Q_{N-m} can be transformed into a valid MIN. Since Q_N is made of 4^m Q_{N-m} 's, there are m links bypassing every Q_{N-m} (see Figure 6.2 for an

example where $m = 2$). In Q_N , one row of $4^m \cdot d_m(N - m)$ $m \times m$ switches can be introduced to encapsulate these links. The principle is illustrated in Figure 6.10. The shadowed switches are dummy switches permanently set to *straight*. Only switches in the lowest row participate in routing. If all switches are counted, the network cost (number of switches) is $c_m(N) = \frac{N}{m} 4^{N-m}$.

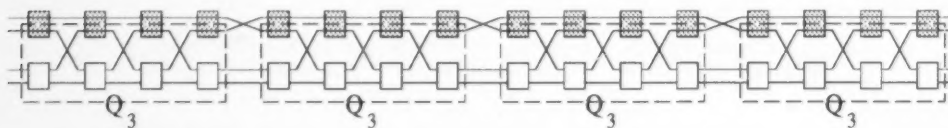


Figure 6.10: Using dummy switches to eliminate links that bypass stages in Q_4 .

2. Figure 6.2 exemplifies that (for $m = 2$) after adding dummy switches, the connection pattern between any 2 adjacent stages is dominated by the identity permutation, except that 2 adjacent links are twisted. The twist may involve 1 or 2 pairs of switches (Figure 6.11). Two alternative remedies, both involve inserting a new stage between every pair of adjacent stages, for removing multiple connections in Figure 6.11 are considered:⁶

- (a) If 2 types of switches are allowed, a dummy stage of m $k \times k$ switches, $k = \frac{N}{m}$, can be inserted between every 2 adjacent stages. A *full-access* pattern (§ 2.4.1) connects the dummy stage to the 2 original stages. The dummy stage, together with the 2 original stages, form a Clos rearrangeable network [24], so any permutation can be performed (using the algorithm in [93]). The 2 cases in Figure 6.11 can be handled as follows. In Figure 6.12a, all the switches in the dummy stage are set to straight. In Figure 6.12b all switches in the dummy stage are set to straight except 1.
- (b) If only one type of switches is allowed, then a more tedious procedure is needed. We assume that $m \leq k$ (if $m > k$, exchange m and k). Similar to

⁶Sometimes, such as the Q_4 in Figure 6.10, redistributing the links between 2 adjacent Q_3 's in Q_4 eliminates multiple connections without adding dummy stages. We study the general case only.

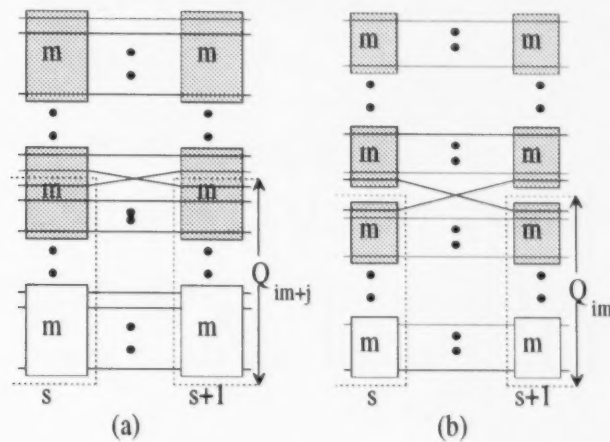


Figure 6.11: The twist in a) involves 2 switches while b) involves 4. i and j are integers, where $i \geq 1$ and $0 < j < m$.

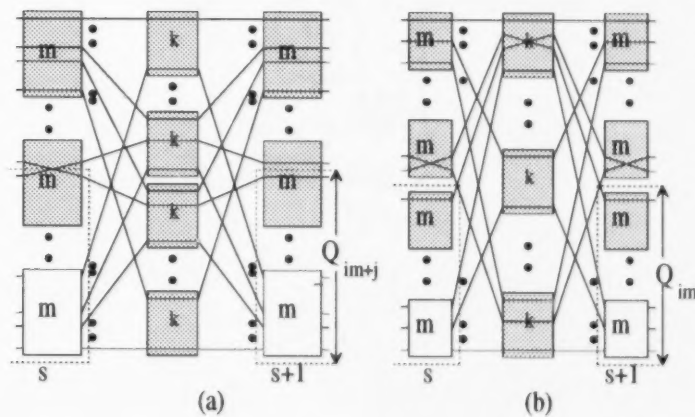


Figure 6.12: Elimination of multiple connections using 2 types of switches.

the previous case, insert a dummy stage of k $m \times m$ switches between every 2 adjacent stages. Since $N \geq m^2$, 2 cases are considered:

$m^2 \leq N < 2m^2$: Number the switches in every stage from 0 to $k-1$ downwards. Number the inputs/outputs of each switch from 0 to $m-1$ downwards. Connect output j of switch i in stage s to input j of switch $(i+j) \bmod k$ in the dummy stage. Connect input j of switch i in stage $s+1$ to output j of switch $(i+j) \bmod k$ in the dummy stage. The above 2 patterns are mirror images of each other. The resulting 3-stage sub-network between stages s and $s+1$ has no multiple connections. Now the 2 cases in Figure 6.11 can be handled as shown in Figure 6.13 (which has $N = 12$ and $m = 3$).

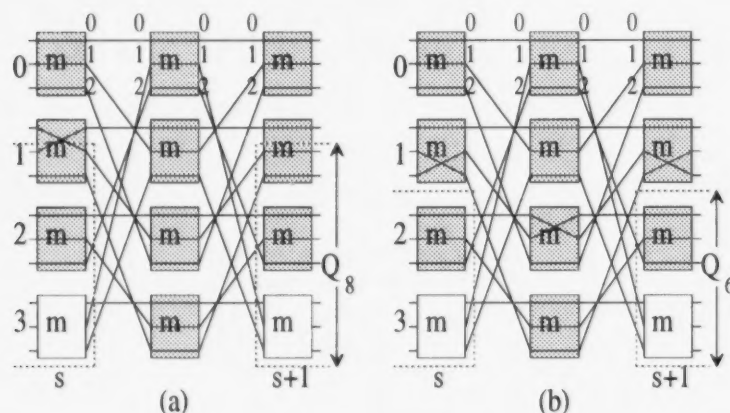


Figure 6.13: Elimination of multiple connections using 1 type of switches.

$N \geq 2m^2$: Arbitrarily select m pairs of switches in stages s and $s+1$ connected by the identity permutation. Use the full-access pattern to connect them to the m corresponding switches in the dummy stage. Set all m switches in the dummy stage to straight such that the identity permutation is realized. Repeat this process until there are $< 2m^2$ pairs of switches. Apply the above procedure for $m^2 \leq N < 2m^2$ to handle the remaining switches that contain the twist.

The depth of Q_N now becomes $d_m(N) = 2 \cdot 4^{N-m} - 1$. If only $m \times m$ switches are used, the cost is $c_m(N) = \frac{N}{m}(2 \cdot 4^{N-m} - 1)$. If $k \times k$ switches are also allowed, the cost measured by the number of cross-points is $c_m(N) = m^2 4^{N-m} + k^2(4^{N-m} - 1)$.

Hence, Q_N can be transformed into a proper MIN by inserting dummy switches. $\square$

The network resulting from Lemma 6.8 case 2a is called the *extended Clos*. Since the wide-sense nonblocking network Q_N can be transformed into a proper MIN, and obviously no wide-sense nonblocking MIN can have width $< N$, we conclude that:

Theorem 6.9 *N is a tight lower bound on the width of wide-sense nonblocking MINs.*

6.8 Advanced Results

In the introduction we raised the question “How many stages of SE are needed to get a wide-sense nonblocking MIN?” The solution is based on Theorem 6.10.

Theorem 6.10 *By serially cascading a finite number of any rearrangeable MINs of width N , a wide-sense nonblocking MIN is obtainable.*

Proof: Without loss of generality, assume every width N rearrangeable MIN ρ has an $m \times m$ switch in its input stage (otherwise, the idea that a Q_N can be obtained from 4 Q_{N-1} ’s allows us to ‘simulate’ large switches using smaller ones). In the extended Clos (Figure 6.14) which has $2 \cdot 4^{N-m} - 1$ stages, every 3 consecutive stages form a Clos rearrangeable network. While all shadowed switches have fixed settings, only the remaining $m \times m$ switches participate in routing. Every sub-network R_i can be replaced by a MIN ρ , whose rearrangeability allows it to perform the fixed (partial) permutation realized by R_i . The $m \times m$ switch in the input stage of ρ becomes the ‘active’ $m \times m$ switch in the first stage of R_i (relabel ρ if the $m \times m$ switch in its input stage is not at the bottom). Now it is clear that 4^{N-m} ρ ’s are needed to have Q_N embedded. $\square$

Corollary 6.2 *At most $(3n - 4)(4^{N-2} - 1) + 1$ stages of SE are needed to construct a wide-sense nonblocking MIN, where $n = \lg N$.*

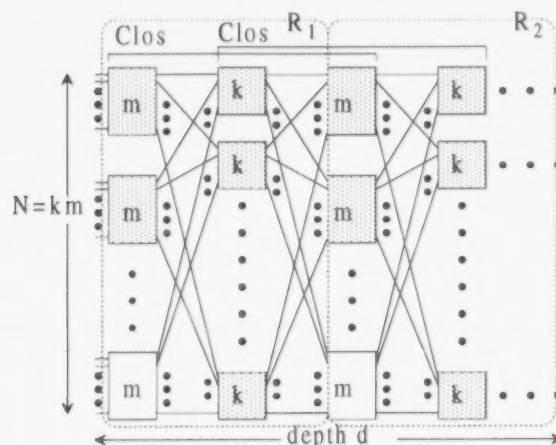
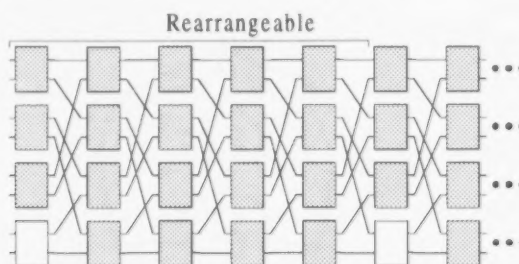


Figure 6.14: An extended Clos network.

Proof: A size N rearrangeable MIN made of $3n-4$ stages of SE [80] can be used as the ρ in Theorem 6.10 (Figure 6.15). (Note that the last stage in Q_N does not require $3n-4$ stages of SE to simulate.) $\square$

Figure 6.15: SE as a building block for width N wide-sense nonblocking MINs.

The set of *universal permutations* U is defined by $U = \{u : \text{a rearrangeable iterated MIN can be built using } u\}$. Similarly, define $W = \{w : \text{a wide-sense nonblocking iterated MIN can be built using } w\}$. Then we have:

Theorem 6.11 $U = W$.

Proof: Since all universal permutations are equivalent to the perfect shuffle [43], Corollary 6.2 implies that $U \subseteq W$. $W \subseteq U$ is also true because every wide-sense nonblocking MIN is rearrangeable. $\square$

Therefore, although a *universal* MIN [43, 110, 118, 119, 129] is defined as one that can simulate a rearrangeable MIN (of a larger depth) using multiple passes, actually it can also simulate a wide-sense nonblocking MIN.

6.9 Concluding Remarks

We introduced a width N multi-path recursive network called the *Quadruplet* (Q) as a vehicle for expressing our routing algorithm. For $N \geq 3$, Q_N was shown to be wide-sense nonblocking. The simplicity of Q 's structure suggests that it has just captured some critical properties of wide-sense nonblocking networks. In addition, Q_N is strictly $(N-2)$ -nonblocking, which is also the highest strict-sense k -nonblockingness any width N MIN can attain. By transforming Q into a proper MIN, we uncovered a fundamental difference between wide- and strict-sense nonblocking MINs: the former can have width N but the latter cannot. By serially cascading a certain number of any rearrangeable MINs of width N , a wide-sense nonblocking MIN can be obtained. It is then shown that at most $(3n-4)(4^{N-2}-1)+1$ stages of SE are needed to attain wide-sense nonblockingness. At the end, a new implication of the well known notion of universality was presented.

Finding the tight lower bound on the width of strictly nonblocking MINs appears to be a challenging problem. Note that Q_N is not a *standard path* (§ 2.3.1) wide-sense nonblocking network because its routing algorithm is not *oblivious*.⁷ A standard path wide-sense nonblocking MIN [69] derived from the crossbar network (Figure 2.4) has width $2N-2$, which is the current minimum. We expect that the tight lower bound on the width of standard path wide-sense nonblocking MINs to be $> N$. This is yet to be proven (lower bounds on the cost of standard path wide-sense nonblocking MINs can be found in [49]). Finding the tight lower bound on the number of SE stages needed to attain wide-sense nonblockingness (or rearrangeability) is difficult because the complexity of the routing algorithm usually increases as the hardware becomes

⁷An oblivious routing algorithm uses solely the source and target addresses to route a request.

more compact.⁸ Moreover, the exercise will not be rewarding because at least $O(N)$ ⁹ stages of SE are needed, resulting in a MIN more expensive than an $N \times N$ crossbar.

Optimizing the cost of $v(m, n, r)$ Clos-type wide-sense nonblocking MINs by making trade-offs between width and depth is another interesting direction. In the extremes, the sufficient depth is $2 \cdot 4^{N-m} - 1$ when the width is N , but the sufficient width is $2n-1$ when the depth is 3 (Theorem 2.1).¹⁰ Does an optimal point exist between the extremes?

⁸[124, p. 108]: "It is reasonable to expect that any saving in hardware gained by improving the number of stages required for rearrangeability will more than be offset by the additional complexity of routing."

⁹In a wide-sense nonblocking MIN of width N , every request must share a switch with every other request (Theorem 6.1). Since any request can share a switch with only $m-1$ other requests at any stage ($m \times m$ switches are used), the depth must be at least $O(N)$.

¹⁰ $\lceil \frac{3n}{2} \rceil$ is the sufficient width when $r = 2$ [10]. In general, width $n + \min(n, r) - 1$ is necessary [69].

Chapter 7

Conclusion

This thesis examines 4 sub-topics related to routing on multi-path, recursively decomposable MINs. Before specific routing problems are addressed, Chapter 2 points out that, in general, permutation routing on arbitrary MINs of depth ≥ 12 is an NP-complete problem. While the problem can be solved in $O(N)$ time for MINs of depth ≤ 3 , the complexity landscape for MINs of depth between 4 and 11 is unknown.

Chapter 3 introduces the k -GCN and presents some of its symmetry properties. We show that by using the 2 alternative paths in a 1-GCN with equal probabilities, the average packet delay can be minimized, regardless of the traffic patterns, under some restrictions. We anticipate that this result can be generalized to a k -GCN, where $1 < k < N$. Identifying the symmetry property necessary for other multi-path MINs to share a similar conclusion is an interesting future direction.

Chapter 4 presents a search-based deterministic algorithm for routing permutations on the ADM network at compile time. By exploring the regularity in the ADM topology, a sequential implementation of the algorithm performed well in our experiment. Empirically, it takes linear time to process an average permutation. An upper bound on the average time complexity is also derived (but no closed-form solution is known). The complexity of the ADM permutation routing problem is likely to lie between $O(N \log N)$ and $O(N)$. After restructuring the search process, the worst-case complexity for our algorithm is shown to be $O(N^3)$.

Chapter 5 introduces the notion of k -nonblocking networks as a framework for interpolating the cost gap between blocking and nonblocking networks with a spectrum of new networks. k -nonblockingness can conveniently quantify the power of a network, especially when it is exposed to unspecified (circuit-switching) traffic. A systematic scheme for constructing k -nonblocking network in the strict sense is given. By giving efficient parallel and sequential routing algorithms, we show that k -nonblocking networks do not require extra hardware/routing overheads.

The main result in Chapter 6 is that a new, multi-path, recursive network called the Quadruplet (Q) is wide-sense nonblocking. By turning Q into a proper MIN of width N , we conclude that strict-sense nonblocking MINs cannot have width N but wide-sense nonblocking MINs can. The number of SE stages sufficient for wide-sense nonblockingness is also derived. A new implication of the notion of universality is uncovered. By merging Tables 5.2 and 6.1, new results presented in Chapters 5 and 6 are summarized in Table 7.1. More research is needed to complete the table.

Minimum depth	Network obtainable (Beneš' classification)	k -nonblocking in the		Examples
		wide sense	strict sense	
1	Blocking, non-full-access	0	0	SE (§ 2.4.2)
$\lg N$	Blocking, full-access	1	1	Ω (§ 2.4.3)
$\lg N + 1^*$	$\vdots$	2	2	ESC (§ 2.4.3)
?	$\vdots$	3	3	?
$2 \lg N - 1$	Rearrangeable	$\vdots$	$\vdots$	Beneš (§ 2.4.4)
?	$\vdots$	$\vdots$	$\vdots$	?
$2 \cdot 4^{N-2} - 1^*$	Wide-sense nonblocking	$N-1, N$	$N-2$	Extended Clos*
∞^*	Strict-sense nonblocking	—	$N-1, N$	None*

Table 7.1: An overview of new results (marked by *) in Chapters 5 and 6.

In general, to achieve the ultimate goal of building a unified theory of routing, more research on the generalization of existing network/routing models is needed. Using recursive network models typically simplifies analysis (e.g., [125]). Designing “generic” routing algorithms that are applicable to a wide range of network models is also a promising direction.

Bibliography

- [1] A. A. Abonamah, F. N. Sibai, and N. K. Sharma. Conflict resolution and fault-free path selection in multicast-connected cube-based networks. *IEEE Transactions on Computers*, 43(3):374–380, March 1994.
- [2] George B. Adams III and Howard Jay Siegel. On the number of permutations performable by the augmented data manipulator network. *IEEE Transactions on Computers*, 31(4):270–277, April 1982.
- [3] D. P. Agrawal. Graph theoretical analysis and design of multistage interconnection networks. *IEEE Transactions on Computers*, pages 637–648, 1983.
- [4] Hamid Ahmadi and Wolfgang E. Denzel. A survey of modern high performance switching techniques. *IEEE Journal on Selected Areas in Communications*, 7(7):1091–1103, September 1989.
- [5] S. B. Akers and B. Krishnamurthy. A group-theoretic model for symmetric interconnection networks. *IEEE Transactions on Computers*, 38:55–567, April 1989.
- [6] G. S. Almasi and A. Gottlieb. Interconnection networks. In *Highly Parallel Computing*, pages 276–299. Benjamin/Cummings Publishing Company Inc., 1989.
- [7] S. Arora, T. Leighton, and B. Maggs. On-line algorithms for path selection in a nonblocking network. In *Proceedings of the Annual ACM Symp. on Theory of Computing*, pages 149–158, 1990.
- [8] Shay Assaf and Eli Upfal. Fault tolerant sorting network. In *Proceedings for the 31st Annual Symposium on Foundations of Computer Science*, volume 1, pages 275–284, 1990.
- [9] P. K. Bansal, Kuldeep Singh, and R. C. Joshi. Routing and path length algorithm for a cost-effective four-tree multi-stage interconnection network. *International Journal of Electronics*, 73(1):107–115, July 1992.
- [10] V. E. Beneš. Heuristic remarks and mathematical problems regarding the theory of connecting systems. *Bell System Technical Journal*, 41:1201–1247, 1962.

- [11] V. E. Beneš. Optimal rearrangeable multi-stage connecting networks. *Bell System Technical Journal*, 43:1641-1656, 1964.
- [12] V. E. Beneš. Permutation groups, complexes, and arrangeable connecting networks. *Bell System Technical Journal*, 43:1619-1640, 1964.
- [13] V. E. Beneš. *Mathematical Theory of Connecting Networks and Telephone Traffic*. Academic Press, New York, 1965.
- [14] V. E. Beneš. Blocking states in connecting networks made of square switches arranged in stages. *Bell System Technical Journal*, 60:511-520, 1981.
- [15] Jean Claude Bermond and Jean Michel Fourneau. Independent connections: An easy characterization of baseline-equivalent multi-stage interconnection networks. *Theoretical Computer Science*, 64:191-201, 1989.
- [16] Jean Claude Bermond, Jean Michel Fourneau, and Alain Jean-Marie. Equivalence of multi-stage interconnection networks. *Information Processing Letters*, 26:45-50, September 1987.
- [17] Jean Claude Bermond, Jean Michel Fourneau, and Alain Jean-Marie. A graph theoretical approach to equivalence of multi-stage interconnection networks. *Discrete Applied Mathematics*, 22(3):201-214, March 1989.
- [18] L. N. Bhuyan. Interconnection networks for parallel and distributed processing. *IEEE Computer*, 20:9-12, June 1987.
- [19] L. N. Bhuyan, Qing Yang, and Dharma P. Argawal. Performance of multi-processor interconnection networks. *IEEE Computer*, pages 25-37, February 1989.
- [20] G. A. De Biase. Interconnection structures and parallel computing. *Advances in Parallel Computing*, 1:241-271, 1990.
- [21] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. Elsevier Science Publishing Co. Inc., North Holland, 1976.
- [22] Hasan Cam and Jose A. B. Fortes. Rearrangeability of shuffle-exchange type networks. In *IEEE Symposium on the Frontiers of Massively Parallel Computations*, pages 303-309, 1990.
- [23] D. G. Cantor. On nonblocking switching networks. *Networks*, pages 367-377, 1972.
- [24] Charles Clos. A study of non-blocking switching networks. *Bell System Technical Journal*, 32:406-424, 1953.
- [25] Lawrence J. Corwin and Robert H. Szezarba. *Multivariable Calculus*. Marcel Dekker Inc., New York and Basel, 1982.

- [26] W. J. Dally. Express cubes: Improving the performance of k -ary n -cube interconnection networks. *IEEE Transactions on Computers*, 40(9):1016–1023, September 1991.
- [27] Nabanita Das, Bhargab B. Bhattacharya, and Jayasree Dattagupta. Hierarchical classification of permutation classes in multistage interconnection networks. *IEEE Transactions on Computers*, 43(12):1439–1444, December 1994.
- [28] J. Dunning Davies. *Mathematical Methods for Mathematicians, Physical Scientists, and Engineers*. Ellis Horwood Ltd., 1982.
- [29] Perter J. Denning and Jeffrey P. Buzen. The operational analysis of queueing network models. *Computing Survey*, 10:225–259, 1978.
- [30] D. M. Dias and M. Kumar. Preventing congestion in multi-stage networks in the presence of hotspots. In *IEEE International Conference on Parallel Processing*, volume 1, pages 9–13, 1989.
- [31] Daniel M. Dias and J. Robert Jump. Analysis and simulation of buffered delta networks. *IEEE Transactions on Computers*, C-30(4):273–282, April 1981.
- [32] B. G. Douglass. Rearrangeable three-stage interconnection networks and their routing properties. *IEEE Transactions on Computers*, 42(5):559–567, May 1993.
- [33] J. Duato. On the design of deadlock-free adaptive routing algorithms for multi-computers: theoretical aspects. In *Proceedings of the 2nd European Conference on Distributed Memory Computing*, pages 234–243, April 1991.
- [34] Ehab Elmallah and Joseph Culberson. Multicommodity flows in simple multi-stage networks. *Networks*, 25(1):19–30, January 1995.
- [35] G. E. Carlsson et. al. Small diameter symmetric networks from linear groups. *IEEE Transactions on Computers*, 41:218–220, February 1992.
- [36] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multi-commodity flow problems. *SIAM Journal of Computing*, 5:691–703, December 1976.
- [37] Mathew Farrens, Brad Wetmore, and Allison Woodruff. Alleviation of tree saturation in multi-stage interconnection networks. In *Proceedings of Supercomputing*, pages 400–409, 1991.
- [38] Paul Feldman, Joel Friedman, and Nicholas Pippenger. Non-blocking networks. In *Proceedings of the Annual ACM Symp. on Theory of Computing*, pages 247–254, 1986.

- [39] Paul Feldman, Joel Friedman, and Nicholas Pippenger. Wide-sense non-blocking networks. *SIAM J. Disc. Math.*, 1(2):158–173, May 1988.
- [40] T. Y. Feng. Data manipulating functions in parallel processors and their implementations. *IEEE Transactions on Computers*, c-23:309–318, March 1977.
- [41] T. Y. Feng. A survey of interconnection networks. *IEEE Computer*, pages 12–27, December 1981.
- [42] Tse-Yun Feng and Seung-Woo Seo. A new routing algorithm for a class of rearrangeable networks. *IEEE Transactions on Computers*, 43(11):1270–1280, November 1994.
- [43] Charles M. Fiduccia and Elaine M. Jacobson. Universal multi-stage networks via linear permutations. In *Proceedings of Supercomputing*, pages 380–389, 1991.
- [44] Michael R. Garey and David S. Johnson. *Computer and Intractability: An Introduction to the Theory of NP-completeness*. W. H. Freeman and Company, 1979.
- [45] Israel Gazit and Mirosław Malek. On the number of permutations performable by extra-stage multi-stage interconnection networks. In *IEEE International Conference on Parallel Processing*, pages 461–471, 1987.
- [46] W. Morven Gentleman. Some complexity results for matrix computations on parallel processors. *Journal of the ACM*, 25(1):112–115, January 1978.
- [47] A. Ghafoor and T. R. Bashkow. A study of odd graphs as fault-tolerant interconnection networks. *IEEE Transactions on Computers*, 40:225–232, February 1991.
- [48] Allan Gottlieb and J. T. Schwartz. Networks and algorithms for very-large-scale parallel computation. *IEEE Computer*, pages 27–36, January 1982.
- [49] L. Halpenny and C. Smyth. Minimal nonblocking standard-path networks. *Electronics Letters*, 28(12):1107–1109, June 1992.
- [50] A. Harissis and A. Ambler. A new multiprocessor interconnection network with wide sense non-blocking capabilities. In *Proceedings of IEEE Midwest Symposium on Circuits and Systems*, pages 923–926, 1989.
- [51] J. Hecquard and R. Acharya. The PSMH: a pyramid of fractional dimension. In *IEEE Symposium on the Frontiers of Massively Parallel Computations*, pages 475–478, 1988.
- [52] Calvin J. A. Hsia and C. Y. Roger Chen. Permutation capability of multi-stage interconnection networks. In *IEEE International Conference on Parallel Processing*, volume 1, pages 338–346, 1990.

- [53] Shing-Tsaan Huang and Satish K. Tripathi. Finite state model and compatibility theory: new analysis tools for permutation networks. *IEEE Transactions on Computers*, 35:591-601, July 1986.
- [54] Joseph Y. Hui. *Switching and Traffic Theory for Integrated Broadband Networks*. Kluwer Academic Publishers, Boston, 1990.
- [55] F. K. Hwang. A mathematical abstraction of the rearrangeability conjecture for shuffle-exchange networks. *Operations Research Letters*, 8:85-87, April 1989.
- [56] K. Hwang and F. A. Briggs. *Computer Architecture and Parallel Processing*. Computer Organization and Architecture. McGraw-Hill, 1985.
- [57] Ching Yuh Jan and A. Yavuz Oruc. Fast self-routing permutation switching on an asymptotically minimum cost network. *IEEE Transactions on Computers*, 42(12):1469-1479, December 1993.
- [58] A. Jean-Marie. Re-routing and resequencing in multi-stage interconnection networks. In *IEEE International Conference on Parallel Processing*, pages 453-460, 1987.
- [59] Hong Jiang, L. N. Bhuyan, and Jogesh K. Muppala. MVAMIN: mean value analysis algorithms for multi-stage interconnection networks. *Journal of Parallel and Distributed Computing*, 12:189-201, 1991.
- [60] A. E. Joel Jr. On permutation switching networks. *Bell System Technical Journal*, 47:813-822, May-June 1968.
- [61] A. E. Joel Jr. Circuit switching: Unique architecture and applications. *IEEE Computer*, 12:10-22, June 1979.
- [62] William H. Kautz, Karl N. Levitt, and Abraham Waksman. Cellular interconnection arrays. *IEEE Transactions on Computers*, 17:443-451, May 1968.
- [63] Kichul Kim and C. S. Raghavendra. A simple algorithm to route arbitrary permutations on 8-input 5-stage shuffle/exchange network. In *Proceedings of the Fifth International Parallel Processing Symposium*, pages 398-403, 1991.
- [64] David G. Kirkpatrick, Maria Klawe, and Nicholas Pippenger. Some graph-coloring theorems with applications to generalized connection networks. *SIAM Journal of Algebraic and Discrete Methods*, 6(4):576-582, October 1985.
- [65] I. Koren and Z. Koren. Discrete and continuous models for the performance of reconfigurable multi-stage systems. *IEEE Transactions on Computers*, 40(9):1024-1033, September 1991.
- [66] C. K. Kothari, S. Lakshmivarahan, and H. Peyravi. A note on rearrangeable networks. Technical report, School of Engineering and Computer Science, University of Oklahoma, November 1983.

- [67] Clyde P. Kruskal and Marc Snir. A unified theory of interconnection network structure. *Theoretical Computer Science*, 48:75-94, 1986.
- [68] V. P. Kumar and S. M. Reddy. Augmented shuffle-exchange multi-stage interconnection networks. *IEEE Computer*, 20:30-40, June 1987.
- [69] C. H. Lam. Unpublished research note, December 1992.
- [70] C. H. Lam. An algorithm for routing permutations on the augmented data manipulator network. *Journal of Computing and Information. Special Issue: Proceedings of the Sixth International Conference on Computing and Information*, 1(1), 1994.
- [71] C. H. Lam and E. Elmallah. Load-balancing on the extra stage generalized cube network. In *Proceedings of the Seventh Annual Canadian High Performance Computing Symposium*, pages 429-436, 1993.
- [72] C. H. Lam and E. Elmallah. On a class of k -nonblocking networks. In *Proceedings of the Eighth Annual Canadian High Performance Computing Symposium*, pages 226-233, 1994.
- [73] Duncan H. Lawrie. Access and alignment of data in an array processor. *IEEE Transactions on Computers*, 24:1145-1155, December 1975.
- [74] K. Y. Lee and H. Yoon. The b-network: a multi-stage interconnection network with backward links. *IEEE Transactions on Computers*, 39:966-969, July 1990.
- [75] Thomson Leighton. Methods for message routing in parallel machines. In *Proceedings of the Annual ACM Symp. on Theory of Computing*, pages 77-96, 1992.
- [76] Mary Diane Palmer Leland. *Properties and Comparison of Multi-stage Interconnection Networks for SIMD Machines*. PhD thesis, University of Wisconsin, December 1983.
- [77] Mary Diane Palmer Leland. On the power of the augmented data manipulator network. In *IEEE International Conference on Parallel Processing*, pages 74-78, 1985.
- [78] G. F. Lev, N. Pippenger, and L. G. Valiant. A fast parallel algorithm for routing in permutation networks. *IEEE Transactions on Computers*, 30:93-100, February 1981.
- [79] G. Lin and N. Pippenger. Parallel algorithms for routing in nonblocking networks. In *Proceedings of the Annual ACM Symp. on Parallel Algorithms and Architectures*, pages 272-277, 1991.

- [80] N. Linial and M. Tarsi. Interpolation between bases and the Shuffle Exchange network. *European Journal of Combinatorics*, 10:29-39, 1989.
- [81] Y. S. Liu and S. Dickey. Simulation and analysis of enhanced switch architectures for interconnection networks in massively parallel shared memory machines. In *IEEE Symposium on the Frontiers of Massively Parallel Computations*, pages 487-490, 1988.
- [82] M. Malek and W. W. Myre. Figures of merit for interconnection networks. In *Workshop on Interconnection Networks for Parallel and Distributed Processing*, pages 74-83, 1980.
- [83] D. E. Marquardt and H. S. AlKhatib. C2MP: a cache-coherent, distributed memory multiprocessor system. In *Super Computer Conference*, pages 466-475, 1989.
- [84] G. M. Masson, G. C. Gingher, and S. Nakamura. A sampler of circuit switching networks. *IEEE Computer*, 12:32-48, June 1979.
- [85] R. J. McMillen and H. J. Siegel. MIMD machine communication using augmented data manipulator network. In *Seventh Annual International Symposium on Computer Architecture*, pages 51-58, May 1980.
- [86] Robert J. McMillen. A survey of interconnection networks. In *IEEE Global Telecommunications Conference*, pages 105-113, 1984.
- [87] Riccardo Melen. A general class of rearrangeable interconnection networks. *IEEE Transactions on Communications*, 39(12):1737-1739, December 1991.
- [88] Bernard L. Menezes. Interconnection networks for fifth-generation computers. In *IEEE Symposium on the Frontiers of Massively Parallel Computations*, pages 503-506, 1988.
- [89] Chris Mitchell and Peter Wild. One-stage one-sided rearrangeable switching networks. *IEEE Transactions on Communications*, 37(1):52-56, January 1989.
- [90] Wayne G. Nation and Howard Jay Siegel. Disjoint path properties of the data manipulator network family. *Journal of Parallel and Distributed Computing*, pages 419-423, September 1990.
- [91] P. Newman. ATM technology for corporate networks. *IEEE Communications Magazine*, 30(4):90-101, April 1992.
- [92] Michael J. O'Donnell and Carl H. Smith. A combinatorial problem concerning processor interconnection networks. *IEEE Transactions on Computers*, C-31(2):163-164, February 1982.

- [93] D. C. Opferman and N. T. Tsao-Wu. On a class of rearrangeable switching networks: I and II. *Bell System Technical Journal*, 50:1579-1618, May 1971.
- [94] D. S. Parker and C. S. Raghavendra. The Gamma network: a multiprocessor network with redundant path. In *Ninth Annual International Symposium on Computer Architecture*, pages 73-80, April 1982.
- [95] D. Scott Parker Jr. Notes on shuffle/exchange-type switching network. *IEEE Transactions on Computers*, 29:213-222, March 1980.
- [96] Janak H. Patel. Performance of processor-memory interconnections for multiprocessors. *IEEE Transactions on Computers*, C-30(10):771-780, October 1981.
- [97] M. C. Pease III. The indirect binary n -cube microprocessor array. *IEEE Transactions on Computers*, 26:458-473, May 1977.
- [98] G. F. Pfister and V. A. Norton. "Hot Spot" contention and combining in multistage interconnection networks. *IEEE Transactions on Computers*, pages 943-948, October 1985.
- [99] N. Pippenger. On rearrangeable and nonblocking switching networks. *Journal of Computer and System Sciences*, 17:145-162, 1978.
- [100] A. Pombortsis, J. Vlahavas, and C. Halatsis. On the performance of packet switching interconnection networks for multiprocessor systems. In *International Conference on Parallel Processing and Applications*, pages 421-428, 1987.
- [101] D. K. Pradhan and K. L. Kodandapani. A uniform representation of single- and multistage interconnection networks used in SIMD machines. *IEEE Transactions on Computers*, C-29:777-791, September 1980.
- [102] C. S. Raghavendra and R. V. Boppana. On self-routing in Beneš and shuffle-exchange networks. *IEEE Transactions on Computers*, 40(9):1057-1064, September 1991.
- [103] C. S. Raghavendra and A. Varma. Rearrangeability of the 5-stage shuffle/exchange network for $n = 8$. In *IEEE International Conference on Parallel Processing*, pages 119-122, 1986.
- [104] D. A. Reed. A simulation study of multi-microcomputer networks. In *IEEE Tutorial: Advanced Computer Architecture*, pages 226-228, 1983.
- [105] D. A. Reed and Dirk C. Grunwald. The performance of multicomputer interconnection networks. *IEEE Computer*, 20:63-73, June 1987.
- [106] Arch D. Robison and Danny Soroker. An algebraic framework for edge-disjoint permutations on hypercubes. In *IEEE Symposium on Parallel and Distributed Processing*, pages 214-221, 1992.

- [107] G. E. Schmidt. The Butterfly parallel processor. In *Proc. of the Second International Conference on Supercomputers*, volume 1, pages 362–365, 1987.
- [108] Robert E. Shannon. *Systems Simulation: the Art and Science*. Prentice-Hall, 1975.
- [109] Dong-Jye Shyy and Chin-Tau Lea. $\log_2(n, m, p)$ strictly nonblocking networks. *IEEE Transactions on Communications*, 39(10):1502–1510, October 1991.
- [110] H. J. Siegel. The universality of various types of SIMD machines interconnection networks. In *Proceedings of the 4th Annual Symposium on Computer Architecture*, pages 70–79, 1977.
- [111] H. J. Siegel. Interconnection networks for SIMD machines. *IEEE Computer*, pages 57–65, June 1979.
- [112] H. J. Siegel. *Interconnection Networks for Large-Scale Parallel Processing*. Computer Organization and Architecture. McGraw-Hill, second edition, 1990.
- [113] H. J. Siegel and W. T. Y. Hsu. Interconnection networks. In *Computer Architecture: Concept and Systems*, pages 225–264, 1988.
- [114] H. J. Siegel, W. G. Nation, C. P. Kruskal, and L. M. Napolitano. Using the multi-stage cube network topology in parallel supercomputers. *Proceedings of IEEE*, 77:1932–1953, December 1989.
- [115] H. J. Siegel and S. D. Smith. Study of multistage SIMD interconnection networks. In *Fifth Annual International Symposium on Computer Architecture*, pages 223–229, April 1978.
- [116] F. Sovis. On rearrangeable networks of the Shuffle Exchange type. *Computers and Artificial Intelligence*, 7(4):359–373, 1988.
- [117] H. S. Stone. Parallel processing with the perfect shuffle. *IEEE Transactions on Computers*, 20:153–161, February 1971.
- [118] T. H. Szymanski and V. C. Hamacher. On the universality of multipath multi-stage interconnection networks. *Journal of Parallel and Distributed Computing*, 7(3):541–569, December 1989.
- [119] Ted H. Szymanski. On the universality of multistage interconnection networks. In *IEEE International Conference on Parallel Processing*, pages 316–323, 1986.
- [120] Ted H. Szymanski. *On Interconnection Networks for Parallel Processors*. PhD thesis, University of Toronto, 1988.
- [121] Fouad A. Tobagi. Fast packet switch architectures for broad-band integrated services digital networks. *Proceedings of IEEE*, 78(1):131–167, January 1990.

- [122] William F. Trench and Bernard Kolman. *Multivariable Calculus with Linear Algebra and Series*. Academic Press, New York and London, 1972.
- [123] Anujan Varma and C. S. Raghavendra. On permutations passable by the Gamma network. *Journal of Parallel and Distributed Computing*, pages 72–91, March 1986.
- [124] Anujan Varma and C. S. Raghavendra. *Interconnection Networks for Multiprocessors and Multicomputers: Theory and Practice*. IEEE Computer Society Press, 1994.
- [125] G. Della Vecchia and C. Sanges. Recursively scalable networks for message passing architectures. In *International Conference on Parallel Processing and Applications*, pages 33–40, 1987.
- [126] Darryl Lawrence Willick. Mean value analysis models for multi-stage interconnection networks. Master's thesis, University of Saskatchewan, November 1990.
- [127] Chuan-Lin Wu and Tse-Yun Feng. Routing techniques for a class of multi-stage interconnection networks. In *IEEE International Conference on Parallel Processing*, pages 197–205, April 1978.
- [128] Chuan-Lin Wu and Tse-Yun Feng. On a class of multistage interconnection networks. *IEEE Transactions on Computers*, c-29(8):694–702, August 1980.
- [129] Chuan-Lin Wu and Tse-Yun Feng. The universality of the shuffle-exchange network. *IEEE Transactions on Computers*, 30:324–331, May 1981.
- [130] Chuan-Lin Wu and M. Lee. Performance analysis of multi-stage interconnection network configurations and operations. *IEEE Transactions on Computers*, 41:18–27, January 1992.
- [131] Y. Yang and G. M. Masson. Nonblocking broadcast switching networks. *IEEE Transactions on Computers*, 40(9):1005–1015, September 1991.
- [132] Yao-Ming Yeh and Tse-Yun Feng. On a class of rearrangeable networks. *IEEE Transactions on Computers*, 41(11):1361–1379, November 1992.
- [133] Hyunsoo Yoon, Kyungsook Y. Lee, and Ming T. Liu. Performance analysis and comparison of packet switching interconnection networks. In *IEEE International Conference on Parallel Processing*, pages 542–545, 1987.
- [134] H. Y. Youn and C. C.-Y. Chen. A comprehensive performance evaluation of crossbar networks. *IEEE Transaction on Parallel and Distributed Systems*, 4(5):481–489, May 1993.

- [135] Abdou Youssef. Off-line permutation scheduling on circuit-switched fixed routing networks. In *IEEE Symposium on the Frontiers of Massively Parallel Computations*, pages 389-396, 1992.
- [136] Abdou Youssef and Bruce Arden. Equivalence between functionality and topology of log n -stage banyan networks. *IEEE Transactions on Computers*, 39:825-832, June 1990.
- [137] Abdou Youssef and Bruce Arden. Topology of efficiently controllable banyan multi-stage networks. *Microprocessing and Microprogramming*, 16(1):3-13, 1992.

Appendix A

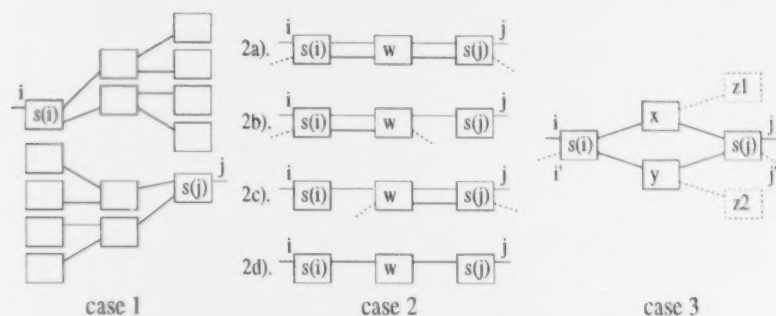
Permutation Routing Complexity: 3-stage INs

To prove theorem A.1, an algorithm that takes $O(N)$ time is given in pseudo code. The algorithm is tedious but yet straight forward so its correctness is evident.

Theorem A.1 : *For a 3-stage MIN, it takes $O(N)$ time to determine whether a given permutation π is admissible, where N is the number of inputs/outputs.*

Proof: Denote the switch that is directly connected to input/output t by $s(t)$. Assume all switches are initialized to a state called *unknown* (U). Given a valid π , it takes $O(1)$ time to identify one of the 3 exclusive cases (Figure A.1) for each request (i, j) , where $j = \pi(i)$, in the permutation:

1. $s(i)$ and $s(j)$ are not connected via any switch in the middle stage: We conclude that the permutation cannot be realized in one pass.
2. $s(i)$ and $s(j)$ are connected via exactly one switch in the middle stage: Four sub-cases are possible as shown in Figure A.1. We present 3 procedures below because cases b) and c) are symmetrical.
 - *Case 2a:* Apply the following $O(1)$ procedure:

Figure A.1: Possible cases for a request (i, j) .

```

if (StateOf( $s(i)$ ) = StateOf( $w$ ) = StateOf( $s(j)$ ) =  $U$ ) then
  /* otherwise, they are properly set already */
  set  $s(i)$  and  $w$  to straight
  set  $s(j)$  according to the request  $(i, j)$ 
endif

```

- *Case 2b:* Apply the following $O(1)$ procedure:

```

if (StateOf( $s(i)$ ) = StateOf( $w$ ) =  $U$ ) then
  set  $s(i)$  to straight
  set  $w$  according to the request  $(i, j)$ 
  setswitch( $s(j)$ ,  $i, j$ )
else
  if ( $s(i)$  and  $w$  are in states compatible with the request  $(i, j)$ ) then
    setswitch( $s(j)$ ,  $i, j$ )
  else
    conclude that the permutation cannot be realized
  endif
endif

```

where the subroutine setswitch() is defined below:

```

proc setswitch( $x$ :switch,  $i$ :input,  $j$ :output)
  if (StateOf( $x$ ) =  $U$ ) then
    set  $x$  according to the request  $(i, j)$ 
  else
    if (StateOf( $x$ ) is compatible with the request  $(i, j)$ ) then
      return
    else
      conclude that the permutation cannot be realized
    endif
  endif
endproc

```

- *Case 2d:* Execute the following $O(1)$ steps:

```

setswitch( $s(i), i, j$ )
setswitch( $s(j), i, j$ )
setswitch( $w, i, j$ )

```

3. $s(i)$ and $s(j)$ are connected via 2 switches in the middle stage: Apply the following $O(1)$ procedure:

```

if (StateOf( $s(i)$ ) = StateOf( $s(j)$ ) =  $U$ ) then
  case ( $\pi(i')$  is associated with output switch) of
    z1: set  $s(i)$  and  $x$  according to the request ( $i', \pi(i')$ )
        setswitch( $y, i, j$ )
        setswitch( $s(j), i, j$ )
    z2: symmetrical to the previous case, skipped
     $s(j)$ : set  $s(i)$  to straight
        set  $s(j)$  according to the request ( $i, j$ )
        setswitch( $x, i, j$ )
        setswitch( $y, i, j$ )
    otherwise: conclude that the permutation cannot be realized
  endcase
else /* StateOf( $s(i)$ )  $\neq U$  or StateOf( $s(j)$ )  $\neq U$  or both */
  if (StateOf( $s(i)$ )  $\neq U$ ) then
    if ( $s(i)$  connects  $i$  to  $x$ ) then
      setswitch( $x, i, j$ )
    else
      setswitch( $y, i, j$ )
    endif
    setswitch( $s(j), i, j$ ) /* using  $x$  or  $y$ , whichever is available */
  else /* StateOf( $s(i)$ ) =  $U$  and StateOf( $s(j)$ )  $\neq U$  */
    if ( $s(j)$  connects  $j$  to  $x$ ) then
      setswitch( $x, i, j$ )
    else
      setswitch( $y, i, j$ )
    endif
    set  $s(i)$  according to request ( $i, j$ ) /* using  $x$  or  $y$ , whichever is available */
  endif
endif

```

Since no loops are involved and every statement takes $O(1)$ time, the entire routing process takes $O(N)$ time. Storage requirement is also $O(N)$. $\square$

Appendix B

Row Data from the Simulation of the 1-GCN

The average queue length at every stage i (denoted Q_i) in the 1-GCN is measured in the simulation experiment and tabulated in Tables B.1 and B.4 for uniform and hotspot traffic, respectively. The average queue length (denoted $\bar{Q}$) of the network, which is the variable to be observed, is tabulated together with the variance (denoted Var.) and the 95% confidence interval (denoted C.I.) in Tables B.2, B.3, B.5 and B.6. $\bar{Q}$ is computed by:

$$\bar{Q} = \frac{1}{4} \sum_{i=1}^4 Q_i,$$

where Q_i is the average queue length in the i th simulation run. Furthermore,

$$Q_i = \frac{1}{n-1} \sum_{j=1}^{n-1} q_{ij},$$

where q_{ij} is the average queue length at stage j in the i th simulation run (we noticed that $\bar{Q}$ still reaches minimum when $p_0 = 0.5$ even if $Q_i = \frac{1}{n+1} \sum_{j=0}^n q_{ij}$). From [108] we extracted the following equation:

$$\text{C.I.}_{95\%} = \bar{Q} \pm t_{0.975} \frac{s}{\sqrt{\mathcal{N}-1}} = \bar{Q} \pm 3.18 \frac{s}{\sqrt{3}},$$

where $t_{0.975} = 3.18$ is the critical value of t in student's t distribution, s is the standard deviation and $\mathcal{N} = 4$ is the sample size. In all tables, C.I. equals $3.18 \frac{s}{\sqrt{3}}$ and w represents the workload factor.

w	p_0	Stages ($N = 8$)				Stages ($N = 16$)				
		3	2	1	0	4	3	2	1	0
6	4	5.1575	211.81	22.405	2.28	3.8575	221.97	20	15.235	2.297
6	5	3.375	3.9675	5.25	4.0625	2.19	3.2775	4.78	5.17	4.61
6	6	4.12	225.13	18.53	2.78	4.06	224.67	14.378	13.685	2.642
5	2	7.25	519.91	19.063	0.8125	4.9225	516.49	15.718	10.125	0.702
5	4	1.31	3.94	3.655	1.6225	1.11	3.97	3.4825	3.9825	1.297
5	5	1.1225	1.595	1.44	1.345	0.8275	1.295	1.36	1	1.42
5	6	0.81	2.9675	3.8425	1.44	1.315	4.2025	4.39	3.545	1.53
5	8	6.1875	518.84	19.935	0.9375	5.97	523.31	17.253	13.283	0.767
4	0	21	600.47	17.967	0.2825	21.645	608.94	18.033	12.83	0.44
4	2	0.905	126.63	15.653	0.655	1.1275	124.16	16.748	14.173	0.64
4	4	0.655	0.72	0.5625	0.75	0.72	1.065	1.125	1	0.797
4	5	0.595	0.6575	0.5925	0.625	0.845	0.7975	1.0475	0.845	0.702
4	6	0.4675	0.655	0.875	0.9075	0.765	1.1075	0.8725	0.875	0.902
4	8	1.0325	119.62	16.75	1.1275	1.0025	130.55	19.515	9.5475	0.877
4	10	24.375	617.69	16.69	0.3775	18.315	604.64	18.423	13.735	0.347
3	0	0.6875	139.72	9.5	0.315	0.8125	130.8	16.89	14	0.345
3	2	0.5025	0.9075	1.065	0.4675	0.485	0.825	0.955	1.1875	0.47
3	4	0.685	0.5925	0.565	0.375	0.47	0.7025	0.44	0.5625	0.595
3	5	0.56	0.375	0.5	0.6225	0.4375	0.4675	0.455	0.4825	0.375
3	6	0.345	0.655	0.4375	0.75	0.56	0.455	0.6575	0.6075	0.56
3	8	0.47	0.9675	0.7825	0.625	0.6075	0.83	1.3725	1.6075	0.5
3	10	0.6875	126.87	20.315	0.41	0.9375	135.14	15.14	15.622	0.362
2	0	0.3475	0.7825	0.685	0.345	0.4225	0.9675	0.7975	0.61	0.252
2	2	0.41	0.31	0.3425	0.185	0.3625	0.485	0.33	0.4875	0.347
2	4	0.345	0.4725	0.3475	0.22	0.4225	0.33	0.44	0.3625	0.357
2	5	0.4375	0.44	0.28	0.185	0.3625	0.405	0.235	0.36	0.36
2	6	0.2825	0.3125	0.44	0.3125	0.4225	0.3125	0.3925	0.2975	0.39
2	8	0.3425	0.5975	0.5925	0.2175	0.375	0.39	0.4675	0.595	0.312
2	10	0.6275	0.655	0.7825	0.41	0.3775	0.625	0.53	0.7325	0.327

Table B.1: Uniform traffic: average queue length (over 4 runs) at each stage.

w	p_0	Q_1	Q_2	Q_3	Q_4	Var.	$\bar{Q}$	C.I.
6	4	125.63	113.185	107.12	122.5	54.21205	117.1088	13.51806
6	5	5.065	5.31	5.56	2.5	1.512905	4.60875	2.258251
6	6	117.375	120	129.5	120.435	20.99271	121.8275	8.412028
5	2	269.13	264.62	270.125	274.065	11.29506	269.485	6.170364
5	4	4.315	6.625	1.685	2.565	3.561069	3.7975	3.464628
5	5	1.065	0.94	1.75	2.315	0.307081	1.5175	1.017403
5	6	1.75	3.25	6.37	2.25	3.222075	3.405	3.295599
5	8	271.435	256.06	274.31	275.75	61.63173	269.3887	14.41347
4	0	309.875	315.38	305.06	306.56	15.68895	309.2188	7.272161
4	2	72.75	63.75	77.56	70.5	24.70755	71.14	9.126018
4	4	0.685	0.56	0.38	0.94	0.041505	0.64125	0.374037
4	5	0.565	0.565	0.685	0.685	0.0036	0.625	0.110158
4	6	0.56	1	0.935	0.565	0.041538	0.765	0.374185
4	8	69.62	63.5	66.5	73.125	12.81282	68.18625	6.571868
4	10	319.185	310.94	324.5	314.13	26.46055	317.1888	9.444217
3	0	65.06	76	86.88	70.5	65.1449	74.61	14.81858
3	2	0.75	0.875	0.88	1.44	0.071342	0.98625	0.490388
3	4	0.44	0.435	0.625	0.815	0.024467	0.57875	0.287183
3	5	0.315	0.56	0.5	0.375	0.009456	0.4375	0.178536
3	6	0.62	0.25	0.685	0.63	0.029867	0.54625	0.317295
3	8	0.75	1.25	0.75	0.75	0.046875	0.875	0.3975
3	10	72.31	74	83.565	64.5	45.98379	73.59375	12.44999
2	0	0.75	1.185	0.56	0.44	0.080092	0.73375	0.519591
2	2	0.185	0.185	0.5	0.435	0.02048	0.32625	0.262741
2	4	0.38	0.315	0.315	0.63	0.016837	0.41	0.238235
2	5	0.375	0.25	0.44	0.375	0.004737	0.36	0.126369
2	6	0.5	0.31	0.315	0.38	0.005867	0.37625	0.140631
2	8	0.565	0.5	0.565	0.75	0.008713	0.595	0.171371
2	10	0.685	0.315	1.25	0.625	0.113792	0.71875	0.619331

Table B.2: Uniform traffic and $N = 8$: average queue length, variance, and confidence interval.

w	p_0	Q_1	Q_2	Q_3	Q_4	Var.	$\bar{Q}$	C.I.
6	4	86.14333	83.23	89.98	83.58667	7.269281	85.735	4.95008
6	5	4.06	3.706667	4.873333	4.996667	0.294008	4.409167	0.99551
6	6	87.29333	82.27333	85.83	81.58333	5.694125	84.245	4.381068
5	2	178.7267	182.77	181.8967	179.71	2.642013	180.7758	2.984242
5	4	3.52	3.333333	4.123333	4.27	0.155269	3.811667	0.723452
5	5	1.206667	1.25	1.373333	1.043333	0.013947	1.218333	0.216825
5	6	4.5	4.5	2.476667	4.706667	0.82788	4.045833	1.670514
5	8	184.69	188.5833	183.8967	181.2933	6.825691	184.6158	4.79667
4	0	210.06	215.7533	215.4167	211.8333	5.785791	213.2658	4.416191
4	2	53.97667	45.56333	49.91667	57.31333	19.38286	51.6925	8.083054
4	4	0.896667	1.063333	1.376667	0.916667	0.036867	1.063333	0.35252
4	5	0.773333	1.02	0.813333	0.98	0.011078	0.896667	0.193238
4	6	0.81	1.143333	0.833333	1.02	0.018869	0.951667	0.252201
4	8	49.02333	54.85333	55.10333	53.83	6.049402	53.2025	4.515675
4	10	211.8767	211.6233	211.1467	214.4167	1.610719	212.2658	2.33011
3	0	58.41667	51.23	53.75	52.18667	7.62178	53.89583	5.068678
3	2	1.063333	1.373333	0.793333	0.726667	0.065085	0.989167	0.468391
3	4	0.54	0.543333	0.753333	0.436667	0.013247	0.568333	0.211314
3	5	0.48	0.54	0.416667	0.436667	0.002236	0.468333	0.086819
3	6	0.48	0.623333	0.543333	0.646667	0.004372	0.573333	0.1214
3	8	1.353333	0.896667	1.603333	1.226667	0.064828	1.27	0.467463
3	10	52.96	55.43667	52.85333	59.95667	8.291247	55.30167	5.2866
2	0	0.79	0.5	1.023333	0.853333	0.035636	0.791667	0.346587
2	2	0.416667	0.606667	0.356667	0.356667	0.010519	0.434167	0.188299
2	4	0.356667	0.44	0.396667	0.316667	0.002102	0.3775	0.084177
2	5	0.293333	0.21	0.54	0.29	0.01535	0.333333	0.227468
2	6	0.293333	0.356667	0.333333	0.353333	0.000635	0.334167	0.04628
2	8	0.396667	0.54	0.376667	0.623333	0.010424	0.484167	0.187452
2	10	0.58	0.686667	0.52	0.73	0.006352	0.629167	0.153082

Table B.3: Uniform traffic and $N = 16$: average queue length, variance, and confidence interval.

w	p_0	Stages ($N = 8$)				Stages ($N = 16$)				
		3	2	1	0	4	3	2	1	0
6	4	4.155	231.78	129.53	148	3.6425	241.38	119.02	132.42	177.4
6	5	2.81	4.875	124.09	224.22	2.6075	4.7	113.61	214.08	176.7
6	6	3	243.94	124.6	152.06	3.4675	234.33	123.13	140.74	174.7
5	2	8.8125	540.12	119.91	12.402	6.61	540.69	117.18	109.98	53.39
5	4	0.905	3.6225	87.25	95.093	0.9675	4.53	86.938	109.53	151.5
5	5	1.0625	1.25	1.935	129.91	0.9375	1.205	2.0625	127.31	175.1
5	6	1.7775	3.75	79.875	93.938	1.2325	4.5325	82.998	115.63	149.4
5	8	6.78	545.75	125.16	7.875	5.4375	531.67	120.34	110.5	49.61
4	0	22.595	626.13	127.72	0.4075	17.75	615.03	116.55	113.58	0.347
4	2	0.94	138.06	125.69	1.405	0.9875	139.27	120.06	112.94	35.04
4	4	0.5	0.625	1.375	17.158	0.78	0.7975	1.22	60.53	104.4
4	5	0.875	0.9375	0.7825	2.625	0.6125	0.6075	0.9675	2.405	148.8
4	6	0.9725	0.8725	1.125	19	0.7975	0.845	1.2825	63.613	113.1
4	8	0.875	147.1	119.81	2	1.14	149.28	118.14	109.88	36.86
4	10	20.375	615.19	122.12	0.315	22.092	618.81	123.69	109.59	0.345
3	0	0.9975	137.75	122.69	0.3475	0.67	140.61	112.25	110.06	0.297
3	2	0.6575	1.2475	3.31	0.785	0.5	1.0175	3.1275	95.188	23.09
3	4	0.44	0.6575	0.53	0.5025	0.4525	0.4975	0.5775	0.8	49.62
3	5	0.5	0.535	0.4375	0.5925	0.56	0.47	0.4075	0.44	41.57
3	6	0.345	0.53	0.53	0.435	0.5	0.7675	0.8275	0.8425	49.55
3	8	0.6225	1.5	2.565	0.81	0.47	0.675	4.58	90.608	25.09
3	10	0.8425	141.47	118.66	0.3475	0.64	143.95	126.47	112.78	0.327
2	0	0.5325	0.6575	1.8125	0.315	0.345	0.7025	0.735	64.72	0.297
2	2	0.3425	0.6225	0.41	0.2825	0.3275	0.28	0.4225	0.6375	1.095
2	4	0.3425	0.3775	0.375	0.345	0.375	0.375	0.345	0.5325	0.657
2	5	0.155	0.185	0.2175	0.4375	0.36	0.33	0.315	0.33	0.36
2	6	0.2825	0.47	0.3475	0.4375	0.33	0.3275	0.3275	0.425	0.282
2	8	0.185	0.47	0.375	0.3775	0.345	0.3775	0.6725	0.53	0.702
2	10	0.345	1.095	1.5925	0.2825	0.53	0.8575	1.28	59.688	0.28

Table B.4: Hotspot traffic: average queue length (over 4 runs) at each stage.

w	p_0	Q_1	Q_2	Q_3	Q_4	Var.	$\bar{Q}$	C.I.
6	4	184.31	179.5	186.81	172	31.87152	180.655	10.36497
6	5	64.25	65.5	62.375	65.81	1.823292	64.48375	2.479103
6	6	188.625	180.315	185.185	182.94	9.30353	184.2663	5.60003
5	2	316.44	338.185	330.685	334.75	68.47466	330.015	15.19258
5	4	43	49.06	44.685	45	4.955392	45.43625	4.087008
5	5	1.935	2.06	1.5	0.875	0.214806	1.5925	0.850922
5	6	41.88	40.56	48.75	36.06	20.69837	41.8125	8.352848
5	8	336.56	338.565	334.815	331.87	6.038731	335.4525	4.511691
4	0	370.5	383.62	373.81	379.75	25.94685	376.92	9.352093
4	2	140.31	129.31	139.62	118.25	80.83852	131.8725	16.50729
4	4	1.25	0.935	0.875	0.94	0.021488	1	0.269128
4	5	0.685	1.065	0.69	1	0.030288	0.86	0.31952
4	6	0.87	0.94	1.065	1.12	0.00978	0.99875	0.181564
4	8	132.25	129.875	127.875	143.815	38.1836	133.4538	11.34501
4	10	376.685	371.75	360.31	365.875	37.85684	368.655	11.29636
3	0	132.065	137.435	129.69	121.685	32.14684	130.2188	10.40964
3	2	3.75	2.62	1.435	1.31	0.982855	2.27875	1.820167
3	4	0.5	0.5	0.75	0.625	0.010742	0.59375	0.190289
3	5	0.815	0.315	0.315	0.5	0.04173	0.48625	0.37505
3	6	0.62	0.56	0.56	0.38	0.0081	0.53	0.165238
3	8	1	2.75	3.63	0.75	1.444419	2.0325	2.206546
3	10	135.25	133.5	125.125	126.375	19.17578	130.0625	8.039759
2	0	2.38	1.125	0.685	0.75	0.465213	1.235	1.252253
2	2	0.56	0.5	0.565	0.44	0.002592	0.51625	0.093476
2	4	0.44	0.315	0.5	0.25	0.009767	0.37625	0.181448
2	5	0.185	0.375	0.06	0.185	0.012667	0.20125	0.206636
2	6	0.44	0.315	0.5	0.38	0.00473	0.40875	0.126265
2	8	0.62	0.25	0.38	0.44	0.017719	0.4225	0.24439
2	10	1.63	0.375	2.685	0.685	0.81333	1.34375	1.655769

Table B.5: Hotspot traffic and $N = 8$: average queue length, variance, and confidence interval.

w	p_0	Q_1	Q_2	Q_3	Q_4	Var.	$\bar{Q}$	C.I.
6	4	163.27	162.48	165.02	166.31	2.23205	164.27	2.742954
6	5	111.3333	111.1233	112.3933	108.3333	2.253019	110.7958	2.755808
6	6	159.8333	173.17	166.4767	164.77	22.79029	166.0625	8.764788
5	2	255.98	261.8333	256.27	249.7067	18.42402	255.9475	7.88059
5	4	63.35333	65.5	68.25	70.89333	8.06718	66.99917	5.214676
5	5	47.9	39.37667	44	42.83	9.264483	43.52667	5.588266
5	6	70	71.39333	65.37667	64.10333	9.315814	67.71833	5.603726
5	8	254.1867	259.1433	251.71	251.6467	9.288281	254.1717	5.595439
4	0	281.56	282.2933	284.44	278.5833	4.397847	281.7192	3.850229
4	2	125.77	129.4167	123.27	117.8933	17.56671	124.0875	7.695054
4	4	19.81	17.79333	23.02333	22.77	4.708647	20.84917	3.983956
4	5	1.146667	1.143333	0.77	2.246667	0.305572	1.326667	1.0149
4	6	23.54333	18.54333	21.75333	23.81333	4.41235	21.91333	3.856572
4	8	127.0433	120.5833	124.6433	130.79	13.74814	125.765	6.807513
4	10	284.54	280.6033	283.2733	287.7067	6.523141	284.0308	4.689158
3	0	118.1233	124.3567	124.8133	116.6033	13.35299	120.9742	6.708967
3	2	31.60333	32.21	33.79333	34.83667	1.632091	33.11083	2.345517
3	4	0.646667	0.54	0.376667	0.936667	0.041625	0.625	0.374579
3	5	0.48	0.313333	0.44	0.523333	0.006147	0.439167	0.14394
3	6	0.666667	0.813333	1.103333	0.666667	0.03178	0.8125	0.327297
3	8	34.10333	36.33667	29.98	27.39667	12.12334	31.95417	6.392602
3	10	129.4167	132.9767	126.3967	122.1467	15.83092	127.7342	7.304989
2	0	21.39333	24.83667	21.73	20.25	2.884774	22.0525	3.118332
2	2	0.476667	0.44	0.456667	0.413333	0.000539	0.446667	0.04262
2	4	0.44	0.52	0.396667	0.313333	0.005574	0.4175	0.137076
2	5	0.21	0.376667	0.376667	0.336667	0.004675	0.325	0.125533
2	6	0.44	0.416667	0.356667	0.226667	0.00685	0.36	0.151954
2	8	0.46	0.52	0.543333	0.583333	0.001994	0.526667	0.081993
2	10	22.89667	19.58	19.60333	20.35333	1.842247	20.60833	2.491956

Table B.6: Hotspot traffic and $N = 16$: average queue length, variance, and confidence interval.

Appendix C

Maple Source Code for Computing $\widetilde{S}_N$

Maple V source code for computing $\widetilde{S}_N$ in chapter 4:

```
fib := proc (N) option remember;
  if N < 2 then
    N
  else
    fib(N - 1) + fib(N - 2)
  fi
end;

phi := proc (B) local N; option remember;
  N := round(B);
  fib(N + 1) + fib(N - 1) - 2
end;

# returns the cardinality of the set  $w^a_b$ .
w := proc(N, a, b) local acc, i; option remember;
  acc := binomial(2^(a - 1), b) * ((N / (2^a))!)^(2*b);
  acc := acc * ((N / (2^(a+1))))!^(2^a-2*b);
  for i from 0 to (a - 2) do
    acc := acc * phi(N / (2^i))^(2^i)
  od;
  acc * (phi(N / (2^(a - 1))))^(2^(a - 1) - b);
end;

# returns the number of leaf nodes in a s level worst-case search tree
t := proc(s) option remember;
  if s = 0 then
```

```

    1
  elif s = 1 then
    4
  else
    (t(s - 1) + t(s - 2)^2)^2
  fi
end;

# returns the number of nodes in a s level worst-case search tree
tau := proc(s) local i, acc; option remember;
  if s <= 0 then
    1
  else
    acc := 0;
    for i from 1 to round(s) do
      acc := acc + t(i)
    od;
    acc
  fi
end;

# returns the number of node-searches for each w(a,b)
NS := proc(a, b, lnN) option remember;
  a - 1 + (tau(lnN - a)^b) * (tau(lnN - a - 1)^2)^(2^(a - 1) - b)
end;

Wall := proc(N, logN) local acc, i, j; option remember;
  acc := 0;
  for i from 1 to (round(logN)-1) do
    for j from 1 to (2^(i-1)) do
      acc := acc + w(N, i, j)
    od;
  od;
  acc
end;

# sn(N) returns the average number of nodes searched for
# each permutation by the ADM permutation routing algorithm.
sn1 := proc(N, logN);
  (logN - 1) * (1 - (Wall(N, logN) / N!))
end;

sn2 := proc(N, logN) local i, j, acc;
  acc := 0;
  for i from 1 to (logN - 1) do
    for j from 1 to (2^(i - 1)) do

```

```
    lprint(i, j, evalf(w(N, i, j)), evalf(NS(i, j, logN)));
    acc := acc + w(N, i, j) * NS(i, j, logN);
  od;
od;
acc / N!
end;

sn := proc(N) local logN;
  logN := round(log[2](N));
  lprint(N, evalf(sn1(N, logN) + sn2(N, logN)));
end;
```